Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Juho Kangas

Opinnäytetyö

Tietokannan konversio

Oracle-pohjaisen tietokannan muunnos PL/SQL-tietokantaohjelmointikielellä

Työn ohjaaja Petri Heliniemi
Tampere 4/2010

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma

Tekijä	Juho Kangas
Työn nimi	Tietokannan konversio Oracle-pohjaisen tietokannan muunnos PL/SQL-tietokantaohjelmointikielellä
Sivumäärä	56
Valmistumisaika	huhtikuu 2010
Työn ohjaaja	Petri Heliniemi

TIIVISTELMÄ

Tässä opinnäytetyössä käsitellään Oracle-tietokantoja, SQL-kieltä ja PL/SQL-tietokantaohjelmointikieltä. Opinnäytetyö pyrkii tarkastelemaan Oracle-pohjaisen tietokannan konversiossa tarvittavia menetelmiä.

Opinnäytetyötä varten luotuun tietokantaan tehtiin määrittely, jonka mukaan se konvertoitiin. Opinnäytetyössä selvitetään, miten konversio onnistui käytännössä ja minkälaisia keinoja Oracle-pohjaisessa tietokannassa siihen oli käytettävissä. Konversiossa käytettiin SQL- ja PL/SQL-kieliä, joilla konversio käytännössä toteutettiin. Opinnäytetyön tarkoitus oli löytää erilaisia keinoja, joita voisi soveltaa tietokantojen hallinnassa ja muuntamisessa.

Konversio tehdään opinnäytetyötä varten luotuun tietokantaan, koska salassapitovaatimusten vuoksi siinä ei voitu kuvata tuotantoympäristössä olevaa tietokantaa. Työ tehtiin täysin itsenäisesti tilannetyyppisenä toteutuksena toimeksiantajan ehdottamasta aiheesta. Koska toimeksiantaja ei halunnut nimeään julki tässä työssä, sitä ei käsitellä opinnäytetyössä tämän enempää. Opinnäytetyö ei sisällä mitään salassa pidettäviä asioita.

Lähteinä käytettiin pääasiassa Oraclen julkaisuja, kuten *SQL Language Reference 11g Release 2* ja PL/*SQL Language Reference 11g Release 2*, jotka ovat SQL- ja PL/SQL-kielten viralliset ohjeistot. Julkaisut ovat vuodelta 2009 ja niitä voi pitää täysin luotettavina ja ajankohtaisina.

Tietokannan luomisessa käytettiin aiempaa kokemusta tuotantotietokantojen rakenteista. Rakenteessa pyrittiin huomioimaan erilaisia tuotantotietokannoissa käytettyjä yleisiä tapoja ja käytäntöjä, jotta tuloksia voitaisiin helpommin soveltaa niihin.

Opinnäytetyön tuloksena on vaiheittainen kuvaus tietokannan konversioprosessista, ja erilaisia ohjenuoria Oracle-tietokannan hallintaan. Näitä tuloksia voidaan soveltaa kaikenlaisten Oracle-pohjaisten tietokantojen muuntamisessa ja kehittämisessä.

Avainsanat Oracle, tietokannat, tietokantaohjelmointi, SQL, PL/SQL, konvertointi

TAMK University of Applied Sciences
Business Information Systems

Writer	Juho Kangas
Thesis	Database conversion Oracle-based database conversion using PL/SQL database programming language
Pages	56
Graduation time	April 2010
Thesis supervisor	Petri Heliniemi

ABSTRACT

This thesis primarily deals with Oracle databases, SQL language and PL/SQL database programming language. The thesis's main aim is to examine different methods in Oracle-based database conversion.

The thesis includes a practical work that first defined the guidelines for conversion and later carried out the conversion. In the thesis are explained how the conversion was made as step by step in practice and what kind of means were available in an Oracle-based database. SQL and PL/SQL languages were used for the practical work of the conversion. The main aim of the thesis was to find different methods and means which could be applied for converting and managing databases.

Initially, the practical work was meant to be of a database in production environment, but because of confidential requirements, the practical work was done completely independently. The thesis subject was given by an employer. Since the employer does not want his name in public, it will not be discussed further. The thesis does not contain any confidential items.

The main sources were Oracle's publications, such as *SQL Language Reference 11g Release 2* and *PL/SQL Language Reference 11g Release 2*, which are SQL and PL/SQL languages official guides. The publications are from year 2009, and they can be considered entirely reliable and current.

The database was created with previous knowledge of the databases in production environment. The database of the practical work was made to correspond to real production databases, so that the results could be more easily applied to them.

The main result of the thesis was a step by step guide of how Oracle database conversion can be accomplished. This guide can be applied to all kinds of Oracle-based databases conversion and development.

Keywords Oracle, databases, database programming, SQL, PL/SQL, conversion

Sisällysluettelo

Käsiteluettelo

Ada	Yhdysvaltain puolustusministeriön kehittämä ohjelmointikieli, joka standardoitiin vuonna 1987
Attribuutti	jonkin ohjelmointiyksikön tietty ominaisuus
Data	informaation tai tiedon raakamuoto
GNU GPL	GNU General Public License eli vapaa ohjelmistolisenssi
Java	ohjelmisto- ja kehitysympäristö, johon sisältyy oliopohjainen ohjelmointikieli ja sen ajoympäristö
Konversio	sisällön tai jonkin rakenteen muuntamista toiseen muotoon
Muuttuja	ohjelmointiyksikkö, jonka arvo voi muuttua
MySQL	vapaan lähdekoodin tietokantatuote
Parametri	komennolle välitettävä tieto
PL/SQL	Procedural Language/Structured Query Language, joka on Oraclen luoma proseduraalinen laajennus SQL-kieleen
Skripti	komentosarja, jonka tietokone tai tietty ohjelma ymmärtää
SQL	Structured Query Language, joka on yleisin käytössä oleva standardoitu tietokantojen hallintakieli
Vakio	ohjelmointiyksikkö, jonka arvo ei muutu

1 Johdanto

Oracle-pohjaiset tietokannat ja järjestelmät ovat tällä hetkellä hyvin yleisessä käytössä yrityksissä. Kenties siksi, että Oracle on ollut 30 vuoden ajan yksi maailman suurimmista tietokantatuotteiden toimittajista. Kaikki käytössä olevat tietokannat ja järjestelmät vaativat aika ajoin jonkinlaista ylläpitoa, ja siksi tietokannan konversio-, muuntamis-, ja kehitystyöt ovat usein hyvin tärkeässä osassa yrityksissä. Tietokannan konversio liittyy usein järjestelmän uudistamisprojektiin. (Oracle 2009b, 1-3.)

Tämän opinnäytetyön keskeisiä asioita ovat Oracle-tietokannat, PL/SQL-tietokantaohjelmointikieli ja tietokannan muunnostyö eli konversio. Alussa tarkastellaan hieman relaatiotietokantoja, SQL-kieltä sekä Oraclea yrityksenä, tietokantana ja hallintajärjestelmänä. Tämän jälkeen keskitytään PL/SQL-kieleen ja tarkastellaan konversiossa tarvittavia asioita esimerkein.

Itse konversio tehdään tätä opinnäytetyötä varten luodusta tilanteesta, jossa tarkastellaan menetelmiä konversion tekemiseen. Siinä käsitellään eräänlaisen terveydenhuollon tietojärjestelmän tietokantaan tehtävää muunnosta. Konversiota ei voitu salassapitovaatimusten vuoksi tehdä tuotantoympäristössä olevasta tietokannasta, joten opinnäytetyössä aihetta lähestytään itsenäisenä tarkasteluna toimeksiantajan ehdottamasta aiheesta. Toimeksiantaja ei halunnut nimeään julki opinnäytetyössä.

Tässä opinnäytetyössä on käytetty lähteinä erilaisia suomenkielisiä tietokantoihin ja SQL-kieleen liittyviä alan julkaisuja, mutta pääasiallisena tietolähteenä on käytetty Oraclen julkaisuja, kuten *SQL Language Reference 11g Release 2* ja *PL/SQL Language Reference 11g Release 2*, joita voi pitää hyvin luotettavina ja kattavina lähteinä. Julkaisut ovat vuodelta 2009 ja niitä päivitetään sitä mukaan, kun kieliä uudistetaan. Aiheeseen johdattelemiseksi on käytetty pienissä määrin myös joitain online-lähteitä, kuten *Tietokone*-lehden artikkeleita ja *Mureakuha*-ohjelmointisivustoa.

Oraclen julkaisujen lähdemerkintöjen sivut on merkitty hieman erikoisella tavalla. Näissä julkaisuissa sivunumerossa kerrotaan kappale ja kyseisen kappaleen sivunumero, joten referenssien lähteet ovat merkitty esimerkiksi *(Oracle 2009a, 13-47 - 13-50)*, joka tarkoittaisi kappaleen 13 sivulta 47 kappaleen 13 sivulle 50. Merkintä *(Oracle 2009a,*

13-47) puolestaan tarkoittaisi kappaleen 13 sivua 47, eikä sivulta 13 sivulle 47.

Lähteistä löytyy myös Ari Hovin julkaisut *SQL-ohjelmointi* vuodelta 2000 ja *SQL-opas* vuodelta 1996, jotka eivät ole yhtä ajankohtaisia Oraclen julkaisuihin verrattuna, mutta ovat hyvin luotettavia ja paikkansapitäviä lähteitä SQL-kieleen. Julkaisuissa on myös hyvää tietoa tietokannoista ja SQL-kielen historiasta. Lisäksi Rovaniemen ammattikorkeakoulun tietokantasovellukset-kurssin materiaalia on käytetty lähteenä. Tämä Maisa Mielikäisen tekemä kurssimateriaali on vuodelta 2007, ja siinä PL/SQL-kieli on esitelty hyvin tiiviisti.

Opinnäytetyön ensimmäisenä tavoitteena on kuvata ymmärrettävästi aihetta, jotta sitä voisi hyödyntää kaikenlaisissa Oracle-pohjaisten tietokantojen muuntamisessa tai kehittämisessä. Lukijalta ei välttämättä vaadita aiempaa tuntemusta Oracle-tietokannoista, mutta suurin mahdollinen hyöty opinnäytetyöstä on juuri niille, jotka ovat jollain tavalla tekemisissä Oracle-tietokantojen kanssa.

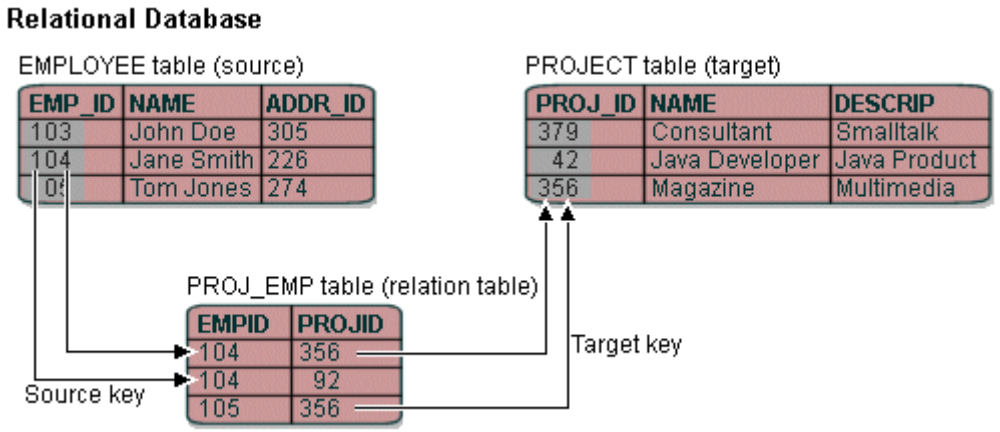
Opinnäytetyön toisena tavoitteena on kuvata konversiossa käytettäviä keinoja siten, että niitä voitaisiin hyödyntää myös muihin tietokantoihin. Opinnäytetyön tietokantaan on koottu samanlaisia rakenteita, jotka toistuvat tuotantotietokannoissa, jotta konvertoitavaan tietokantaan tehtyjä toimenpiteitä voitaisiin helpommin soveltaa niissä. Tavoitteena on myös luoda ohjenuora konversioprosessista, jossa konversion eri vaiheet tulee kuvattua.

2 Relaatiotietokanta

Usein tietokanta ymmärretään jonkinlaisena monimutkaisena järjestelmänä, jonka toteuttamiseen tarvitaan tehokkaat palvelimet ja laitteet, mutta yksinkertaisesti ajateltuna se voi olla vain joukko tietoja. Se voi olla esimerkiksi kokoelma tekstiä tekstitiedostossa tai Excel-taulukko. Relaatiotietokannalla on kuitenkin hieman yksityiskohtaisempi määrittely, vaikka siitä puhuttaessa usein käytetäänkin vain nimitystä tietokanta. Myös puhuttaessa tietokannoista, tarkoitetaan sillä usein juuri relaatiotietokantaa, koska se on nykyisin yleisin käytössä oleva tietokantamalli digitaalisen datan säilytykseen. Kaikki uusimmat tietokannat, kuten Oracle, MySQL, Microsoft SQL Server, Microsoft Access tai PostgreSQL, perustuvat relaatiotietokantamalliin. (Hernandez 2000, 3 - 5; Mureakuha 2006.)

Relaatiotietokanta on tekstitiedostoihin ja Excel-tiedostoihin verrattuna laajempi tietokantajärjestelmä, jossa voidaan käsitellä suuria tietomääriä helposti. Relaatiotietokannan relaatio-sana tulee siitä, että tieto liittyy jollain tavalla toiseen. Kahden eri tiedon välissä on niin sanottuja relaatioita. Relaatiotietokannassa tietoja voidaan yhdistellä toisiinsa ilman, että suurenkaan tietomäärän ylläpitäminen aiheuttaa ongelmia. Tekstitiedostossa tai Excel-taulukossa tällaisen tiedon ylläpitäminen olisi paljon vaikeampaa. (Mureakuha 2006.)

Relaatiotietokantamallin kehitti IBM:n tutkija Edgar F. Codd 1960-luvun lopulla. Codd halusi kehittää uusia tapoja käsitellä suuria tietomääriä, koska oli tyytymätön senaikaisiin tietokantamalleihin. Relaatiotietokantamalli perustuu joukkoteoriaan, matematiikan oppeihin ja tietorakenteiden käyttöön. Relaatiotietokantamallissa tiedoilla on tiedon yksilöivä arvo, joka poikkeaa muista tiedoista. Sitä kutsutaan taulun *pääavaimeksi*. Toiset tiedot voivat viitata näihin yksilöiviin arvoihin taulussa olevilla *viiteavaimilla*, kuten kuvassa 1. (Hernandez 2000, 11 - 17.)



Kuva 1. Relaatiotietokannan tiedot viittaavat yksilöityihin arvoihin (Oracle 2006a)

Tiedot tallennetaan tietokannan tauluihin, joissa on *kenttiä* ja *tietueita* (eli rivejä), jotka voivat viitata toisiin kenttiin ja niiden tietueisiin. Itse taulut eivät sinänsä liity millään tavalla toisiinsa, vaan nimenomaan niiden sisältämien kenttien tietueet. Tiedot siis löytävät toisensa sijaitsevatpa ne fyysisesti missä tahansa. (Hernandez 2000, 11 - 20, 42.)

Fyysisesti relaatiotietokanta sijaitsee tietokoneen kovalevyllä tiedostoina, mutta sellaisessa muodossa, jota käyttäjä ei yleensäkään ymmärrä. Relaatiotietokantajärjestelmä kokoaa nämä tiedot kovalevyltä, ja näyttää ne loogisena kokonaisuutena tauluina ja kenttinä yhteyksineen, jonka käyttäjä näkee tietokantana. (Oracle 2009b, 12-1.)

Relaatiotietokannassa voi olla monen tyyppisiä viittauksia. Viittaus voi olla yhden suhde yhteen, yhden suhde moneen ja monen suhde moneen. Viittauksen tyyppi riippuu siitä, miten tietokanta on toteutettu. Taulussa, jossa on viiteavain toisen taulun kenttään, on yhden suhde moneen -viittaus ja taulussa, jossa viiteavain on samalla pääavain, joka viittaa toisen taulun pääavaimeen, on kyseessä yhden suhde yhteen -viittaus. Jos kahden taulun välillä on yhteystaulu, on kyseessä monen suhde moneen -viittaus. Kuvan 1 esimerkki on monen suhde moneen -viittaus. (Oracle 2006a.)

Relaatiotietokannan yksi tarkoitus on pitää huolta siitä, että viittaukset ovat eheitä ja todella olemassa. *Pääavaimet* ja *viiteavaimet* määritetään tietokantaan, jolloin tietokanta alkaa pitää huolta siitä, että viittaavan *taulun kentän* arvo on sellainen, joka varmasti löytyy viitattavan *taulun kentästä*. Mikäli näin ei tapahdu, antaa relaatiotietokanta virheilmoituksen, että viite-eheyden rikkovaa toimintoa ei voida suorittaa. Tietokanta huo-

lehtii näin siitä, että kaikki viittaukset pysyvät eheinä. (Hovi 1996, 7.)

Tällä tavoin relaatiotietokanta on eheämpi sekä joustavampi ylläpitää kuin vanhoihin tietokantamalleihin perustuvat tietokannat, jotka olivat riippuvaisia enemmänkin fyysisestä toteutuksesta. Aikaisempia tietokantamalleja tässä opinnäytetyössä ei käsitellä, joten tässä opinnäytetyössä termeillä kanta ja tietokanta tarkoitetaan juuri relaatiotietokantaa. (Hernandez 2000, 3 - 20.)

Relaatiotietokanta ei yksinään ole kovinkaan tehokas työväline. Se tarvitsee informaation tallentamiseen ja käsittelemiseen oman hallinnointikielen, jonka käyttö sellaisenaan voi olla hyvin kankeaa. Siksi tietokanta yleensä luodaan jonkinlaisen järjestelmän alustaksi, jolloin järjestelmä käsittelee itse tietokantaa. Sinne on ohjelmoitu kaikki tietokannan käsittelemiseen tarvittavat toiminnot, jolloin käyttäjä voi keskittyä tiedon hallitsemiseen ja tallentamiseen järjestelmän kautta.

3 Structured Query Language (SQL)

SQL-kieli on yleisin käytössä oleva standardoitu tietokantojen hallintakieli. SQL-kielen avulla tietokannasta voidaan hakea tietoa, tallentaa tietoa tai tehdä muita tietokannan hallintaan liittyviä asioita, kuten poistaa, luoda tai muokata tauluja. Lähes kaikki tietokantatuotteet perustuvat SQL:ään, lisäten siihen omat laajennuksensa. Vaikka SQL-kieli on standardoitu, on eri tuotteiden syntakseissa paljon eroja, joita ei tässä opinnäytetyössä kuitenkaan käsitellä. (Hovi 2000, 9 - 11.)

IBM kehitti SQL-kielen 1970-luvun alussa Edgar F. Coddin kehittämän relaatiotietokantamallin hallinnointikieleksi. Aikaisemmin kieltä kutsuttiin nimeltä *SEQUEL (Structured English Query Language)*, mutta se nimettiin myöhemmin SQL:ksi. 1970-luvun loppupuolella nykyinen Oracle kehitti Coddin relaatiotietokantamallin ja SQL:n pohjalta järjestelmiä päämääränään myydä niitä CIA:lle, Yhdysvaltain laivastolle ja muille hallinnollisille elimille. (Hovi 2000, 9 - 11.)

Yleisesti ottaen SQL-kielessä puhutaan komennoista, kyselyistä ja lauseista. Tässä opinnäytetyössä komennolla tarkoitetaan tiettyä kokonaista SQL-komentoa, oli se sitten hakemista tai ylläpitoa. Kyselyllä taas tarkoitetaan sitä, että tietokannasta haetaan tietoa määriteltyjen ehtojen mukaan ja lauseella tai lausekkeella tarkoitetaan yhtä SQL-komennossa olevaa kokonaisuutta käsittävää osaa. Muualla tämä määrittely ei välttämättä täysin päde, sillä käsitteitä käytetään monesti ristiin.

3.1 Yksinkertainen SQL

SQL-kieli on ulkoasultaan melko helpon näköistä, ja yksinkertaisen SQL-kyselyn voi jopa asiaan perehtymätönkin helposti ymmärtää. Esimerkiksi kuvassa 2 kysely hakee *firstname*-kentän tiedot *person*-taulusta.

```
SELECT firstname FROM person; -- Yhden rivin mittaiset kommentit näin
                             /* Useamman rivin
                                kommentit näin */
```

Kuva 2. Yksinkertainen SQL-kysely

Isolla kirjoitetut tekstit ovat SQL-kielen käsky-sanoja (tästä edespäin kaikki SQL- ja PL/SQL-kielten käsky-sanat esitetään SQL-komennoissa isoilla kirjaimilla) ja kaksi väliviivaa merkitsee lopun rivistä kommentiksi sekä riviä pidemmät kommentit merkitään /*-merkinnällä alkavaksi ja */-merkinnällä loppuvaksi. Kaikki SQL-kyselyt lopetetaan puolipisteeseen tai omalla rivillä olevaan kauttaviivaan. (Hovi 2000, 14.)

SQL-kyselyn perusrakenne koostuu *select*-, *from*-, *where*-, *order by* -lauseista. *Select*-lauseella kerrotaan, mitkä sarakkeet haetaan. *From*-lause kertoo, mistä tauluista haetaan. *Where*-lause määrittää hakujen valintaperusteet eli hakuehdot, ja viimeisenä tuleva *order by* kertoo, minkä kenttien mukaan tulosjoukko ryhmitellään (*asc* nousevaan ja *desc* laskevaan järjestykseen). (Hovi 2000, 15.)

SQL-kyselyssä *select*- ja *from*-lauseet ovat pakollisia ja muut lauseet valinnaisia. Nämä SQL-lausekkeet ovat yleisimpiä ja niillä selviää melko pitkälle tietoa haettaessa kannasta. Seuraavassa alaluvussa esitellään tarkemmin SQL:n käyttöä esimerkein. (Hernandez 2000, 15 - 16; Hovi 2000, 15.)

3.2 SQL-Komentoja

Tässä alaluvussa esitellään yleisimmät SQL-lauseet, joita tarvitaan tietoa haettaessa ja tietokantaa muokattaessa, sekä esitellään SQL:n käyttöä erilaisissa tilanteissa. Läheskään kaikkia SQL-kielen komentoja tai käyttötapoja ei kuitenkaan käydä läpi, vaan tarkastellaan yleisimpiä tietokannan peruskäyttämiseen tarvittavia komentoja.

Select-lause

Select-lauseella määritetään tauluista haettavat kentät pilkulla eroteltuna. Haettavan kentän edessä voi olla taulun *aliasnimi*, jota kutsutaan joissain tapauksissa termillä *korrelaationimi*. *Aliasnimi* annetaan Oraclen tuotteissa *from*-lauseessa taulun nimen jälkeen, ja sen käyttö haettavan kentän edessä on suositeltavaa ainakin silloin, kun haetaan useasta taulusta, ja pakollista silloin, kun haetaan samannimisiä kenttiä useasta taulusta. (Oracle 2009a, 19-4.)

Kuvassa 3 on esimerkki *select*-lauseesta, jossa kahdesta eri taulusta haetaan samannimistä kenttää. Tästä johtuen tauluille on annettu *aliasnimet*, jotka *select*-lauseessa viittaavat haettavan taulun kenttään.

```
SELECT p.name, pr.name -- Select-lause (p ja pr ovat aliasnimiä)
FROM person p, project pr;
```

Kuva 3. *Select*-lause ja taulujen aliasnimet SQL-kyselyssä

Where-lause

Where-lausetta käytetään eri komennoissa määrittämään halutut rivit, joille tehdään jotakin. *Where*-lause on komennon rajaava lause. Sitä voidaan käyttää muun muassa *select*-, *delete*-, ja *update*-lauseiden yhteydessä rajaamaan, mitä haetaan, poistetaan tai päivitetään. Ehtoja voidaan putkittaa *and*- ja *or*-sanoilla sen mukaan, halutaanko *where*-lauseen rajaavan kaikkien (*and*) ehtojen vai ainoastaan jonkun (*or*) ehdon täyttyessä. (Oracle 2009a, 19-25.)

Where-ehdoissa voidaan käyttää erilaisia operaattoreita, kuten pienempi kuin (<), suurempi kuin (>) tai yhtäsuuri (=) ja sen sisällä voi olla myös alikyselyitä. Alikyselyä käytetään usein *in*-määreen kanssa, jolla haluttu arvo voi olla joku sen sisällä olevista arvoista. Myös *like*-operaattori on usein käytetty *where*-lauseessa. Sillä voidaan rajata haku esimerkiksi selvittämään, sisältääkö hakutulos tietyn kirjaimen tai vaikka tietyn kirjainyhdistelmän. *Like*-operaattorissa alaviiva (_) tarkoittaa yhtä mitä tahansa merkkiä ja prosenttimerkki (%) yhtä tai useampaa mitä tahansa merkkiä. Kuvan 4 esimerkissä havainnollistetaan näiden operaattoreiden käyttöä *where*-lauseessa. (Oracle 2009a, 7-1 - 7-16.)

```
/* Kysely hakee kaikkien kaupunkien nimet, jotka alkavat K- tai H-kirjaimella
   ja jotka sijaitsevat Pirkanmaalla tai Satakunnassa */
SELECT
    tw.name
FROM
    town tw
WHERE
    (tw.name LIKE 'K%'
    OR tw.name LIKE 'H%')
    AND tw.province_id IN (
        /* Alikysely where-ehdon sisällä */
        SELECT
            pr.province_id
        FROM
            province pr
        WHERE
            pr.name IN ('Pirkanmaa', 'Satakunta')
        );
```

Kuva 4. *Where*-lauseen erilaisia käyttötapoja

Union

Union-määrellä voidaan yhdistää useita kyselyitä yhdeksi. Käytännössä tämä tehdään lisäämällä kokonaisten kyselyjen väliin *union*-määre, kuten kuvassa 5.

```
SELECT
    ci.name,
    ci.population -- Haetaan kaupinkien nimet ja väkiluku
FROM
    city ci
WHERE
    ci.province_name = 'Pirkanmaa' -- Haetaan ne kaupungit, jotka ovat Pirkanmaalla
UNION -- Yhdistetään kaupunkien ja kuntien haku yhteen kyselyyn
SELECT
    tw.name,
    tw.population -- Haetaan kuntien nimet ja väkiluku
FROM
    town tw
WHERE
    tw.province_name = 'Pirkanmaa' -- Haetaan ne kunnat, jotka ovat Pirkanmaalla
ORDER BY
    2 DESC; -- Ryhmitellään laskevaan järjestykseen väkiluvun mukaan
```

Kuva 5. Kaksi SQL-kyselyä yhdistetty *union*-määreellä

Union-määrettä käytettäessä tulee ottaa huomioon, että kaikissa *unioneilla* yhdistetyissä yksittäisissä kyselyissä täytyy hakea yhtä monta kenttää ja kaikkien kenttien tietotyyppien täytyy olla sama. Esimerkiksi *name*- ja *population*-kentän järjestyksen muuttaminen alemmassa kyselyssä aiheuttaisi virheilmoituksen, sillä *name*-kenttä on tekstityyppistä ja *population* on numeerista tietoa. (Oracle 2009a, 9-8 - 9-10.)

Commit ja Rollback

Ennen kuin siirrymme tietokannan hallintaan liittyviin komentoihin, on hyvä tarkastella Oracle-tietokannoissa hyvin oleellisessa asemassa olevia *commit*- ja *rollback*-komen-

toja ja tietokannan *transaktioita*.

Transaktio on sarja komentoja, joita käyttäjä tekee tietokantaan. *Commit* tarkoittaa, että kyseisessä *transaktiossa* tehdyt muutokset tulevat voimaan ja *rollback* vastaavasti kumoaa kaikki *transaktion* muutokset. *Commit*- ja r*ollback*-komennoilla voidaan kumota tai hyväksyä kaikki *Data Manipulation Language* (*DML*) -komennot, mutta *Data Definition Language* (*DDL*) -komennot astuvat voimaan heti, eikä niihin *commit*- tai *rollback*-komennot vaikuta. *DML*-komennot tarkoittavat tietojen muokkaamiseen, lisäämiseen ja hakemiseen liittyviä komentoja, kuten *select*-, *insert*-, *update*- ja *delete*-komentoja. *DDL*-komennot puolestaan tarkoittavat tietokannan rakenteiden lisäämiseen, muokkaamiseen ja poistamiseen tarkoitettuja komentoja, kuten *create table* -, *alter table* - ja *drop table* -komentoja. (Oracle 2009a, 10-2 - 10-3, 13-47, 18-97.)

Transaktion ensisijainen tehtävä on huolehtia siitä, että usea tietokannan käyttäjä voi hallita samoja tietoja samaan aikaan ilman, että siitä aiheutuu ongelmia. Oracle käsittelee *transaktioita* omina kokonaisuuksinaan, jolloin jokaisella käyttäjällä on oma *transaktionsa*. Näin tieto ei pääse korruptoitumaan monien käyttäjien yhtäaikaisesta tietojen hallinnasta. Käyttäjän yksittäinen *transaktio* joko onnistuu tai epäonnistuu, mutta se ei vaikuta muihin käyttäjiin. (Oracle 2009a, 13-47 - 13-50.)

Vaikka käyttäjä tekisi tietokantaan muutoksia, mutta jättää *commit*-komennon antamatta, kumoutuvat kaikki käyttäjän tekemät muutokset (automaattinen *rollback*) ulos kirjautuessa. Tämä on asia, joka täytyy ottaa huomioon Oraclessa. Usein muiden tietokantatuotteiden käyttäjät huomaavat joutuvansa enemmän tai vähemmän ongelmiin Oraclea käyttäessään, koska esimerkiksi MySQL:ssä on oletuksena ns. *autocommit*, joka hyväksyy muutokset heti sekä käsittelee *transaktioita* muutenkin hieman eri tavalla. (Oracle 2009a, 13-47 - 13-50.)

Insert

Insert-komennolla luodaan tauluun uusi rivi. Ensin lauseessa kerrotaan, mihin taulun kenttiin arvoja sijoitetaan, ja sen jälkeen *values*-sanan jälkeen kerrotaan sijoitettavat arvot. Jos kentän tietotyyppi on numeerinen, voidaan arvo merkitä ilman heittomerkkejä. Tietotyyppejä tarkastellaan tarkemmin PL/SQL-kielen yhteydessä. Kuvassa 6 on esimerkki *insert*-komennosta. (Oracle 2009a, 18-54.)

```
/* Luodaan projekti,
   jonka pääavain on 123 ja
   nimi Järjestelmäuudistus */
INSERT INTO project (project_id, pname)
    VALUES (123, 'Järjestelmäuudistus');
COMMIT; -- Hyväksytään transaktioon tehdyt muutokset
```

Kuva 6. *Insert*-komento ja transaktion hyväksyminen

Kaikkiin *not null* -tyyppisiin kenttiin täytyy määrittää arvot riviä luotaessa. *Null* tarkoittaa relaatiotietokannassa tuntematonta arvoa. *Not null* -tyyppiseen kenttään vaaditaan aina arvo, eikä se voi olla tyhjä. Tietyntyyppiset tiedot määritetään *not null* -tyyppisiksi, esimerkiksi pääavain (automaattisesti *not null*) tai sosiaaliturvatunnus henkilö-taulussa voisi olla *not null* -tyyppinen. (Oracle 2009a, 18-54.)

Update

Update-lauseella päivitetään olemassa olevien rivien tietoja. *Update*-lauseessa ensimmäisenä määritetään taulu, jonka tietoja halutaan muuttaa. Sen jälkeen *set*-määreen jälkeen määritetään kaikki ne kentät, joita muutetaan. *Update*-komennon kanssa käytetään usein *where*-lausetta, jolla rajataan ne rivit, joita halutaan muuttaa. (Oracle 2009a, 19-71.)

```
UPDATE city
    SET
        population = '211350',
        tax = '19'
    WHERE name = 'Tampere';
COMMIT;
```

Kuva 7. *Update*-komento ja transaktion hyväksyminen

Delete ja drop table

Delete-komennolla poistetaan rivejä taulusta. Komennossa kerrotaan, mistä taulusta poistetaan, ja *where*-lauseessa määritetään, mitä rivejä poistetaan. Kuvan 8 *delete*-komennossa poistetaan *town*-taulusta rivi, jonka *name*-kentän sisältö on Kuru. *Delete*-komentoa voidaan käyttää myös ilman *where*-ehtoa, mutta silloin kaikki rivit tuhoutuvat taulusta. Lopuksi *transaktio* hyväksytään *commit*-komennolla. (Oracle 2009a, 17-26.)

```
/* Where-lauseessa voidaan määrittää mitkä rivit poistetaan */
DELETE FROM town
    WHERE name = 'Kuru';
COMMIT; -- Hyväksytään transaktioon tehdyt muutokset
```

Kuva 8. *Delete*-komento ja *where*-lause

Kuvan 9 komennossa poistetaan kaikki rivit taulusta *town*, sillä komennossa ei anneta *where*-ehtoa ollenkaan. Tämä komento ei kuitenkaan poista itse taulua, vaan mikäli halutaan poistaa koko taulu, joudutaan käyttämään *drop table* -komentoa, joka tuhoaa koko taulun (kuva 10). (Oracle 2009a, 17-26, 18-5.)

```
/* Komento poistaa kaikki rivit taulusta */
DELETE FROM town;
COMMIT; -- Hyväksytään transaktioon tehdyt muutokset
```

Kuva 9. *Delete*-komento ilman *where*-lausetta

```
/* Taulun tuhoaminen */
DROP TABLE town;
```

Kuva 10. *Drop table* -komento poistaa taulun tietokannasta

Create table

Create table -komennolla luodaan taulu. Komennossa annetaan muun muassa luotavan taulun nimi, kentät, niiden tietotyypit, pääavaimet ja viiteavaimet. Viiteavaimia määritettäessä tulee komennossa kertoa, mihin tauluun, ja mihin kenttään viiteavain viittaa. Pääavaimet ja viiteavaimet voidaan vaihtoehtoisesti määrittää *alter table* -komennolla, joka esitellään alempana, kun taulu on luotu. (Oracle 2009a, 16-6; 12-2.)

Kuvassa 11 on esimerkki *person*-nimisen taulun luomisesta. Esimerkissä kentät, tietotyypit, kentän ominaisuudet ja avaimet määritetään omilla riveillään. Esimerkissä ensimmäisenä kenttänä määritetään *person_id*, jonka tietotyyppi on *number(12)*, ja joka on *not null* -tyyppinen sekä määritetty pääavaimeksi. Tietotyypin jälkeen suluissa oleva numero osoittaa, kuinka pitkiä numeroja kenttää voidaan tallentaa. Alempana *constraint*-määreen jälkeen määritetään vierasavain *hometown_id*-kenttään viittaamaan *town*-taulun *town_id*-kenttään. Vierasavaimen looginen nimi on *fk_person_town*, jota tarvitaan ainoastaan, kun vierasavain-määritystä halutaan muuttaa tai poistaa kokonaan.

```
CREATE TABLE person(
  person_id               NUMBER(12)     NOT NULL PRIMARY KEY,
  name                    VARCHAR2(255)  NOT NULL,
  age                     VARCHAR2(255),
  hometown_id             NUMBER(12)     NOT NULL,
  CONSTRAINT
    fk person_town FOREIGN KEY (hometown_id)
  REFERENCES
    town(town_id));
```

Kuva 11. *Create table* -komennolla luodaan taulu

Kenttien tietotyypit määritettään sen mukaan, minkälaista tietoa kenttään tallennetaan. Esimerkiksi *person*-taulun *name*-kenttään tallennetaan henkilön nimi, jolloin sen tietotyypiksi sopii *varchar2*. *Varchar2* on Oraclessa merkkijonotyyppinen tietotyyppi, eli *varchar2*-tyyppiseen kenttään voidaan tallentaa kirjaimia, numeroita, välilyöntejä tai mitä tahansa merkkejä. *Number*-tietotyyppi on numerotyyppinen, johon voidaan tallentaa ainoastaan numeroita ja *date*-tietotyyppiseen kenttään ainoastaan päivämäärä- ja aika-tietoa. (Oracle 2009a, 16-6, 3-1.)

Eri tietokantatuotteissa on erilaisia tietotyyppejä ja niitä on pelkästään Oracle-pohjaisissa tietokannoissa kymmeniä. Muun muassa geometriatiedolle on Oraclessa olemassa oma tietotyyppinsä. Kuitenkin *varchar2*, *number* ja *date* ovat Oraclen yleisimpiä tietotyyppejä, ja ainoastaan niitä tullaan käyttämään opinnäytetyössä esiteltävässä konversiossa.

Alter table

Alter table -komennolla muokataan olemassa olevia tauluja. Komennolla voidaan pääasiassa tehdä samat asiat, kuin *create table* -komennolla taulua luotaessa, mutta ainoastaan jo olemassa olevaan tauluun. Komennolla voidaan muun muassa poistaa, lisätä tai muokata taulun kenttiä. Hyvin yleinen tapa on käyttää *alter table* -komentoa pää- ja viiteavaimien luomiseen taulun luomisen jälkeen. Kuvassa 12 on esimerkki *alter table* -komennosta, jossa *person*-tauluun luodaan uusi kenttä *project_id*, joka on viiteavain ja viittaa *project*-taulun *project_id*-kenttään. (Oracle 2009a, 12-2.)

```
ALTER TABLE person
ADD project id  NUMBER(12)
ADD CONSTRAINT
   fk person_project_id FOREIGN KEY (project_id)
REFERENCES
   project(project_id);
```

Kuva 12. Alter table -komennolla muokataan tauluja

4 Oracle

Oracle on noin 30 vuoden ajan ollut maailman johtava tietokantateknologian toimittaja, ja lisäksi se on maailman toiseksi suurin itsenäinen ohjelmistoyritys. Oraclen pääalaa ovat tietokantatuotteet, mutta se toimittaa myös muita liike-elämän sovelluksia, sovelluskehittimiä ja päätöksenteon aputyökaluja. Oraclen tunnetuimpia tuotteita ovat Oracle-tietokanta ja tietokannan hallintajärjestelmä, joita kumpaakin voidaan kutsua nimellä Oracle. (Oracle 2009b, 1-3.)

Oraclen perusti sen tämänhetkinen omistaja ja toimitusjohtaja Larry Ellison, joka innostui Edgar F. Coddin relaatiotietokantamallista ja päätti perustaa oman yhtiön nimeltä *Software Development Laboratories* vuonna 1977. Myöhemmin yhtiö nimettiin Oracleksi. Yhtiön ensimmäinen tuote oli *Oracle V2*, joka oli ensimmäinen kaupallinen relaatiotietokannan hallintajärjestelmä. Siitä lähtien Oracle-pohjaiset tietokannat ovat olleet yksi kilpailukykyisimmistä tietokantatuotteista. (Oracle 2009b, 1-3.)

Oraclen uusin aluevaltaus on Sun Microsystems, jonka Oracle 20. huhtikuuta 2009 ilmoitti ostavansa. Kauppa ei kuitenkaan sujunut helposti, ja se meni Euroopan Unionin kilpailuviranomaisten tutkittavaksi, koska kaupan pelättiin rajoittavan kilpailua. Euroopan Unioni kuitenkin hyväksyi kaupan, ja näin Oraclen omistukseen siirtyi tunnettu ohjelmointikieli Java ja vapaalla avoimen lähdekoodin *GNU GPL* -lisenssillä saatava suomalaislähtöinen MySQL. (Tietokone 2010.)

4.1 Oracle tietokantana

Kirjoitushetkellä Oraclen uusin versio tietokannasta on *Oracle Database 11*g *Release 2*, joka julkaistiin 1. syyskuuta 2009. Se tukee muun muassa useita *Windows*-, *Linux*-, *Solaris*- ja *Mac*-pohjaisia käyttöjärjestelmiä. Oraclessa *system*- eli pääkäyttäjän oikeuksilla *v$transportable_platform*-taulusta löytyvät kaikki käyttöjärjestelmät, joihin tietokannan taulualueita voidaan muuntaa. Kuvassa 13 on esitetty *v$transportable_platform*-taulun sisältö ilmaisessa *Oracle Database 10g Express Edition* -versiossa. Kyseinen Oracle-versio siis tukee ainakin näitä kuvassa 13 esiintyviä käyttöjärjestelmiä. (Oracle 2009c, 13-31 - 13-32.)

```
SELECT * FROM v$transportable_platform ORDER BY 2;
```

Query... x

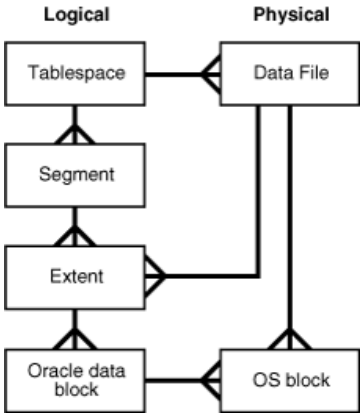
All Rows Fetched: 17 in 0,031 seconds

	PLATFORM_ID	PLATFORM_NAME	ENDIAN_FORMAT
1	6	AIX-Based Systems (64-bit)	Big
2	16	Apple Mac OS	Big
3	15	HP Open VMS	Little
4	5	HP Tru64 UNIX	Little
5	4	HP-UX IA (64-bit)	Big
6	3	HP-UX (64-bit)	Big
7	18	IBM Power Based Linux	Big
8	9	IBM zSeries Based Linux	Big
9	10	Linux IA (32-bit)	Little
10	11	Linux IA (64-bit)	Little
11	13	Linux 64-bit for AMD	Little
12	7	Microsoft Windows IA (32-bit)	Little
13	8	Microsoft Windows IA (64-bit)	Little
14	12	Microsoft Windows 64-bit for AMD	Little
15	17	Solaris Operating System (x86)	Little
16	1	Solaris[tm] OE (32-bit)	Big
17	2	Solaris[tm] OE (64-bit)	Big

Kuva 13. *v$transportable_platform*-taulun sisältö *10g Express Edition* -versiossa

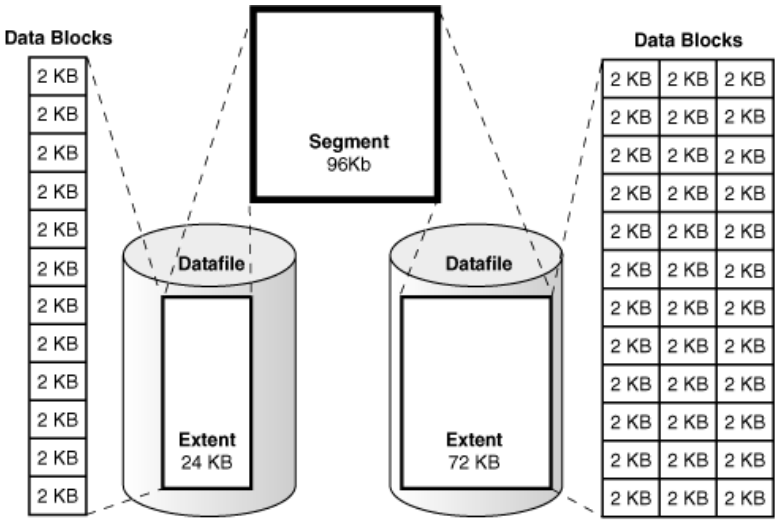
Tietokantana Oracle toimii kahdella eri tasolla - fyysisellä ja loogisella. Fyysisesti tiedot tallennetaan tiedostoihin (*database datafiles*) kovalevylle ja vastaavasti kaikelle tälle tiedolle on oma paikkansa loogisella tasolla. Loogisen tason yksiköitä ovat taulualueet (*tablespace*), segmentit (*segment*), tilavaraukset (*extent*) ja tietoalkiot (*data block*), joista tietoalkiot ovat pienimpiä. (Oracle 2009b, 12-1 - 12-5, 11-1 - 11-9.)

Taulualueet koostuvat yhdestä tai useammasta segmentistä, segmentit yhdestä tai useammasta tilavarauksesta ja tilavaraukset yhdestä tai useammasta tietoalkiosta. Vastaavasti fyysisen tason tiedostot koostuvat yhdestä tai useammasta käyttöjärjestelmän varaamasta tilayksiköstä kovalevyllä. Kuva 14 havainnollistaa loogisen ja fyysisen tason yksiköt ja niiden yhteydet toisiinsa. (Oracle 2009b, 12-1 - 12-5, 11-1 - 11-9.)



Kuva 14. Loogisen ja fyysisen tason yhteydet toisiinsa (Oracle 2009b, 12-1)

Nämä loogisen tason taulualueet ovat taas suurimpia ja niissä olevat taulut ovat se, minkä käyttäjä näkee tietokantana. Kuva 15 havainnollistaa, miten segmentit (*segment*), tilavaraukset (*extent*) ja tietoalkiot (*data block*) koostuvat toisiinsa nähden.



Kuva 15. Segmentit, tilavaraukset ja tietoalkiot toisiinsa nähden (Oracle 2009b, 12-2)

4.2 Oracle tietokannan hallintajärjestelmänä

Tietokannan hallintajärjestelmä huolehtii viime kädessä siitä, että käyttäjällä on mahdollisuus hallita tietokannan taulualueita ja niissä olevien taulujen tietoja. Se on ohjelmisto, joka hallinnoi tietovarastoa, sen organisointia ja tiedon hakemista. Tietokannan hallintajärjestelmään kuuluu Oraclen (2009b, 1-2 - 1-13.) mukaan:

- ydinkoodi (*kernel code*), joka hallinnoi muistia ja tietovarastoa
- metatietoja, eli niin sanotusti tietoja tiedoista, joita Oracle käyttää apuna tietojen

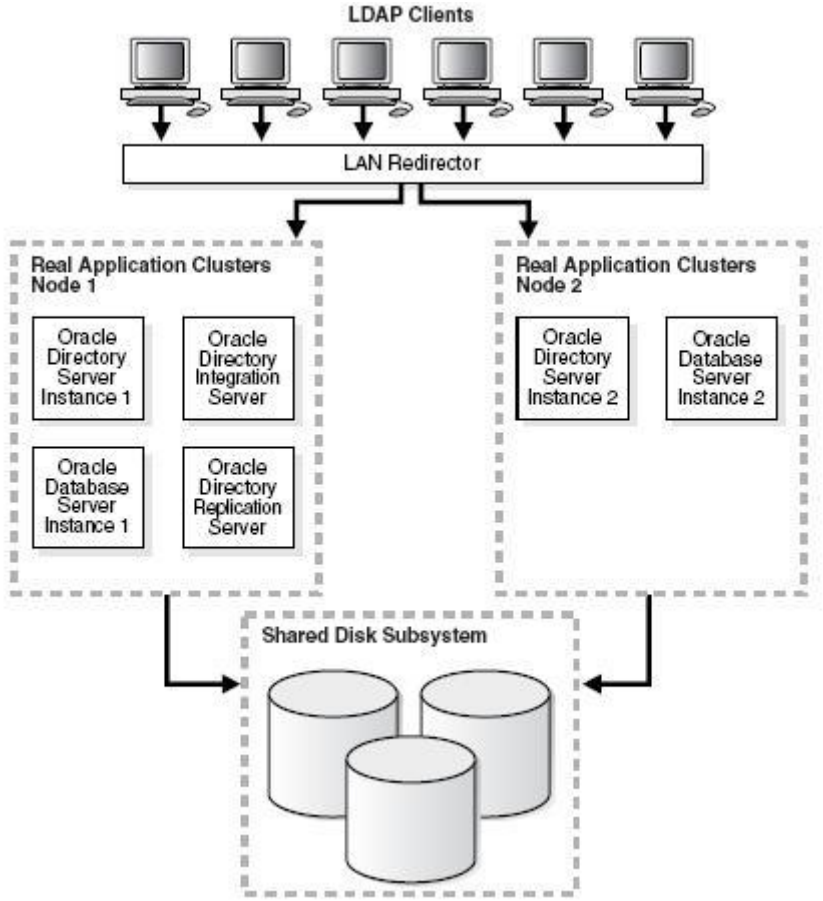
hallinnassa (esimerkiksi *system*-taulualue)
- tuki kyselykielelle, jolla käyttäjä pääsee tietoihin käsiksi (SQL, PL/SQL, Java).

Palvelinprosessi, jossa tietokantapalvelua ylläpidetään, huolehtii tietokannan hallinnoinnista. Asiakasprosessit puolestaan ovat päätelaitteiden prosesseja, tarkoituksenaan lähettää sovelluskoodia tietokantaan tai hallita sitä tietokantasovelluksen avulla. Palvelinprosessi kommunikoi asiakasprosessien kanssa ja mahdollistaa päätelaitteiden pääsyn tietokantaan. Päätelaitteet tarvitsevat ainoastaan asiakasprosessin, jolla ne pääsevät tietokantaan käsiksi palvelinprosessin kautta esimerkiksi pelkällä tietokantatyökalulla. (Oracle 2009b, 1-2 - 1-13.)

Oracle Real Application Clusters (RAC)

Oraclella on tietokannan yhtäaikaista käyttämistä helpottava tietokantaohjelmiston klusterointi-ominaisuus, joka parantaa tietokannan kuormantasausta, skaalautuvuutta ja vikasietoisuutta. Tämän ominaisuuden avulla tietokanta niin sanotusti klusteroidaan eli luodaan monesta eri tietokantaa hallinnoivasta palvelimesta (*node*) yksi kokonaisuus (*cluster*). Tätä ryhmää kutsutaan ryppääksi, ja se tarkoittaa sitä, että mikäli yksi tietokantaa operoiva palvelin kaatuu, jatkaa toinen palvelin tietokannan operoimista, eikä käyttäjä huomaa eroa (kuvassa 16). (Oracle 2006b.)

Kaikki tietokantaa hallinnoivat palvelimet hallinnoivat yhtä ja samaa tiedon säilytyspaikkaa (*storage array*), joten tietokanta on täysin sama riippumatta siitä, mikä yksittäinen palvelin sitä käsitteleekin. Ryppäitä voi olla useampia ja niissä useita eri *noodeja*. (Oracle 2006b.)



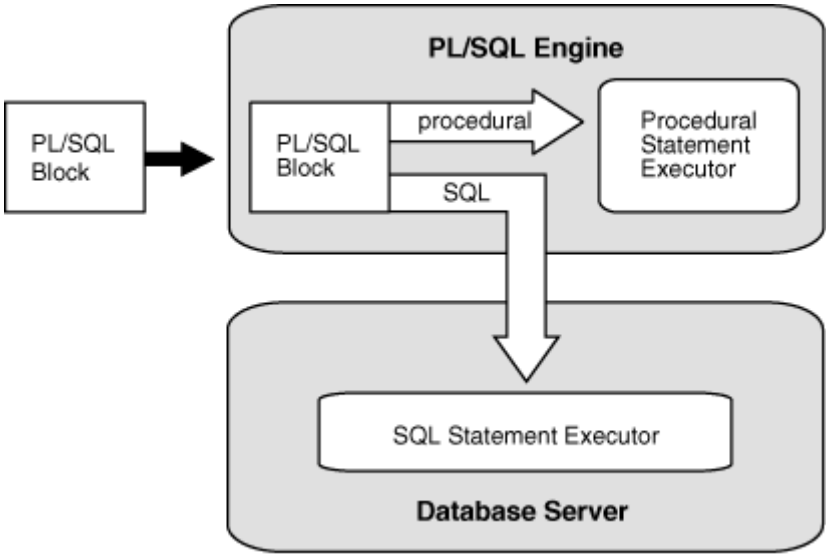
Kuva 16. Oracle RAC:n toimintaperiaate (Oracle 2006b)

4.3 PL/SQL-tietokantaohjelmointikieli

Tässä alaluvussa käsitellään PL/SQL-tietokantaohjelmointikielelle keskeisiä piirteitä, joista osaa käytetään myöhemmin konversiota käsittelevässä luvussa.

Procedural Language/Structured Query Language eli PL/SQL on Oraclen luoma proseduraalinen laajennus SQL-kieleen. PL/SQL-kieli luotiin käsittelemään SQL-käskyjä saumattomasti ja dynaamisesti. Se muistuttaa *Ada*-nimistä ohjelmointikieltä ja sitä käytetään muun muassa tietokantaoperaatioiden, proseduurien, funktioiden ja uusimmissa versioissa myös oliotekniikoiden luomiseen tietokantaympäristössä. PL/SQL on proseduraalinen ohjelmointikieli, ja se luo SQL-kielen ympärille ohjelmointiympäristön, joka mahdollistaa tietokantaoperaatioiden ohjelmoinnin ja normaalia SQL-kieltä tehokkaamman tietokannan hallinnan. (Oracle 2009d 1-1 - 1-9; Kettunen 2000.)

PL/SQL-ohjelmointikieli suoritetaan PL/SQL-moottorissa ja siihen sisältyvät SQL-komennot lähetetään SQL-tulkille, joka käsittelee itse tietokantaa. PL/SQL-moottori näin käsittelee dynaamisesti määriteltyjä tehtäviä SQL-tulkki apunaan (kuva 17). (Oracle 2009d 1-11.)



Kuva 17. PL/SQL-moottori (Oracle 2009d, 1-11)

PL/SQL-kielessä on mistä tahansa ohjelmointikielestä tutut kontrollirakenteet (*if, else, while, for* ja *loop),* joilla voidaan joustavasti yhdessä SQL-kielen kanssa hallita tietokantaa. Myös muuttujien ja poikkeuksien käyttö on mahdollista. Kokonaisuutena koodi koostuu lohkoista (*PL/SQL Block*), joita on kolmea eri tyyppiä. Nämä lohkotyypit ovat *Anonymous, Procedure* ja *Function*. Kontrollirakenteet, muuttujat ja lohkotyypit esitellään alempana. (Oracle 2009d, 1-1 - 1-9; Kettunen 2000.)

Pääasiassa kieltä käytetään Oraclen ympäristössä, johon se on alun perin suunniteltu, mutta sitä voi hyödyntää myös muualla, kuten esimerkiksi joissain vapaan lähdekoodin tietokantatuotteissa. PL/SQL-koodi voidaan tallentaa suoraan tietokantaan, ja sovellus voi suorittaa näitä ohjelmakoodeja suoraan sieltä, jolloin sen ei ensin tarvitse lähettää PL/SQL-lohkoja tietokantaan. Näin ollen tietokannan ja päätteen välisen liikenteen vähentyessä tietokannan päälle rakennettujen sovellusten tietoturva ja suorituskyky paranee. (Oracle 2009b, 8-2 - 8-6.)

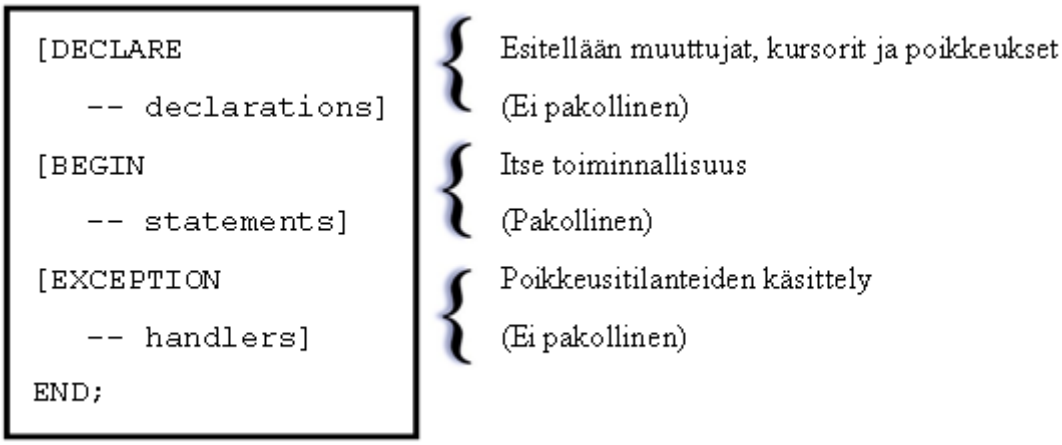
PL/SQL-lohkot tukevat *Data Manipulation Language* (DML) -kieltä, eli taulun tietojen

muokkaamiseen, luomiseen ja hakemiseen tarkoitettuja komentoja, mutta ne eivät tue *Data Definition Language* (DDL) -kieltä, jolla esimerkiksi luodaan, muokataan tai poistetaan tauluja. Käytännössä tämä tarkoittaa sitä, ettei lohkon sisällä voi esimerkiksi poistaa tai luoda tauluja. (Oracle 2009d, 8-27; Mielikäinen 2007a.)

4.3.1 Lohkon rakenne

PL/SQL-lohkot ovat kielen perusyksiköitä. Ne ovat loogisia ohjelmia, joista sovellukset koostuvat. Lohkot voivat sisältää myös alilohkoja. Sovelluksessa voidaan kutsua näitä pieniä loogisia ohjelmia, joissa haluttu toiminnallisuus tapahtuu. Lohkolla on olemassa perusrakenne, joka on esitetty kuvassa 18, jota kaikki kolme eri lohkotyyppiä noudattavat. (Oracle 2009d, 1-5.)

Pääasiassa lohkot ovat joko proseduureja (*Procedure*) tai funktioita (*Function*), mutta niiden lisäksi on olemassa myös anonyymi lohko (*Anonymous*), joka on nimeämätön täysin itsenäinen ja riippumaton lohko. Proseduureja ja funktioita kutsutaan aliohjelmiksi, jotka ovat useimmiten tietokantaan tallennettuja ohjelman osia. Näitä kutsutaan sovelluksessa niiden nimellä. Anonyymia lohkoa ei nimetä ja se voidaan ymmärtää kertakäyttöisenä lohkona, joka tehdään tiettyyn tarkoitukseen suoritettavaksi. Kaikilla lohkoilla on kuitenkin sama seuraavanlainen perusrakenne. (Mielikäinen 2007a.)



Kuva 18. PL/SQL-lohkon rakenne (Oracle 2009d, 1-5)

4.3.2 Tietotyypit ja muuttujat

PL/SQL-kielessä kaikilla muuttujilla, vakioilla ja parametreilla on oma tietotyyppinsä. Tietotyyppi kertoo, minkälaisen arvon muuttuja, vakio tai parametri voi saada. Se määrittää myös, minkälaisia operaatioita tiedolle voidaan tehdä. Esimerkiksi kaksi numerotyypistä (*number*) muuttujaa voidaan laskea yhteen tai ne voidaan kertoa, ja kaksi merkkijonotyyppistä (*varchar2*) muuttujaa voidaan yhdistää peräkkäin. Muuttujan tietotyyppi annetaan muuttujan nimen jälkeen esimerkiksi kirjoittamalla *v_muuttuja varchar2(25)*, joka määrittää 25 merkin pituisen *varchar2*-tyyppisen muuttujan nimeltä *v_muuttuja*. Mikäli muuttujaan halutaan tallentaa tietoa suoraan taulun kentästä, voidaan muuttujan tyyppi määrittää *taulu.kenttä%TYPE*-merkinnällä kyseisen kentän tietotyyppiin sopivaksi. PL/SQL-kielessä muuttujille ja vakioille annetaan arvo *:=* (kaksoispiste ja yhtä kuin) -merkinnällä. (Oracle 2009d, 3-1; Mielikäinen 2007b.)

Tietotyypit jakautuvat neljään pääkategoriaan: yksiarvoiset- (*scalar*), yhdistelmä- (*composite*), viittaavat- (*reference*) ja multimediatietotyypit (*large object* eli *LOB*). Näistä yksiarvoiset tietotyypit ovat yleisimpiä ja niihin kuuluu alakategorioita, jotka ovat *number*-, *character*-, *boolean*-, *datetime*- ja *interval*-tyyppiset muuttujat. (Oracle 2009d, 3-1.)

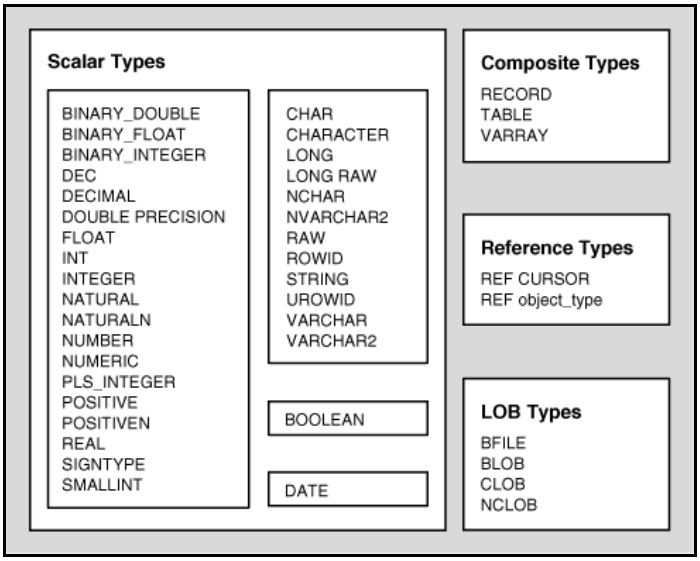
Yksiarvoisten alakategorioiden tarkoituksena on pyrkimys mahdollistaa tietotyyppien yhteensopivuus muihin ohjelmiin tai ohjelmointikieliin. Esimerkiksi kokonaislukutyyppiset (*integer*) tai desimaalilukutyyppiset (*decimal*) muuttujat kuuluvat yksiarvoisten muuttujien *number*-alakategoriaan. Alakategorioita voidaan suoraan käyttää muuttujien tietotyyppeinä, jolloin ne voivat saada minkä tahansa arvon, joka sopii kyseiseen alakategoriaan. Esimerkiksi *number* sopii kaikkiin numerotietoihin. (Oracle 2009d, 3-2; Mielikäinen 2007b.)

Yhdistelmä eli *composite*-tietotyypit ovat sellaisia tietotyyppejä, jotka sisältävät useita eri komponentteja, joilla voi olla eri tietotyyppi. Tällainen tietotyyppi voi olla esimerkiksi taulukko tai yksi SQL-kyselyn tuottama rivi. Siinä on siis useita tietoja, joilla on eri tietotyyppi, ja joita voidaan käsitellä erikseen. Kirjoittamalla *taulu%rowtype* muuttujan jälkeen, voidaan määrittää muuttujan tietotyypiksi taulun tietueeksi sopiva muuttuja. Esimerkiksi komento *row_person person%rowtype;* määrittää *row_person*-muuttujan

tietotyypiksi sellaisen, johon voidaan tallentaa *person*-taulun kokonainen rivi. (Oracle 2009d, 2-16; Mielikäinen 2007b.)

Viittaavat tietotyypit ovat tietotyyppejä, jotka viittaavat johonkin olemassa olevaan tietoon. Hyvänä esimerkkinä ovat kursorit (*cursor*), jotka viittaavat kyselyn tuottamaan riviin yksi kerrallaan. (Oracle 2009d, 6-25.) Kursoreita käsitellään tarkemmin myöhemmin.

Multimedia eli *large object* -tietotyypit ovat isoja tietotyyppejä, joihin voidaan tallentaa raakadataa, kuten kuvia, videoita, äänitteitä ja todella pitkiä tekstejä. Kuvassa 19 on esitetty kaikki Oraclen muuttujatyypit kategorioittain. (Oracle 2009d, 3-21 - 3-26; Mielikäinen 2007b.)



Kuva 19. Oraclen tietotyypit (Mielikäinen 2007b)

4.3.3 Kontrollirakenteet

Kontrollirakenteet ovat PL/SQL-kielen oleellisin laajennus SQL-kieleen. Niiden avulla SQL-kielestä saadaan dynaamisempaa. Kontrollirakenteita on olemassa kolmea eri kategoriaa: ehtorakenteet, silmukkarakenteet ja jaksottaiset rakenteet. Ehtorakenteissa tutkitaan jotain tiettyä ehtoa, jonka täyttyessä tehdään jokin haluttu toimenpide. Ehtorakenteita ovat *if-then-*, *if-then-else-*, *if-then-elsif-* ja *case*-lauseet. (Oracle 2009d, 4-1.)

Silmukkarakenteissa toistetaan haluttua toimenpidettä, kunnes tietty ehto täyttyy. Silmukkarakenteita ovat *loop*-, *while loop* -, *for loop* - ja *cursor for loop* -lauseet. Jaksottaisia rakenteita ovat puolestaan *goto*- ja *null*-lauseet. Niillä voidaan hallita koodin etenemistä jaksottaisesti esimerkiksi merkkaamalla tietty kohta koodissa ja määrittämällä *goto*-määreellä, mistä kohdasta sinne hypätään. (Oracle 2009d, 4-9.)

Ehtorakenteista *if-then*-lauseessa *if*-sanan jälkeen määritetään ehto, jonka täyttyessä siirrytään *then*-osioon, jossa määritettään halutut toimenpiteet. Mikäli ehto ei täyty, ei *then*-osioon siirrytä ollenkaan. Rakenteessa voidaan käyttää lopussa *else*-määrettä, johon siirrytään, mikäli *if*-lauseen ehto ei toteudu. *Elsif*-lauseella voidaan tutkia toteutumattoman *if*-lauseen jälkeen, toteutuuko *elsif*-lauseen ehto. Jos se toteutuu, siirrytään *elsif*-lauseen *then*-osioon. Mikäli aiempi *if*-ehto toteutui, ei *elsif*-lauseeseen siirrytä, vaikka sen ehto toteutuisikin. (Oracle 2009d, 4-1.)

Case-rakenteella voidaan tutkia useita ehtoja kerralla ja määrittää haluttu toiminnallisuus kullekin ehdolle. Yksinkertaisella *case*-rakenteella usein määritetään muuttujalle arvoa tutkimalla toisen muuttujan arvoa. Yksinkertaisuudessaan *case*-lauseessa esitetään *case*-sanan jälkeen muuttuja, jota tutkitaan, ja *when*-sanan jälkeen arvo, jonka toteutuessa suoritetaan *then*-osio. *Case*-rakenteessa voidaan käyttää myös *else*-osiota, johon siirrytään, kun mikään *when*-lause ei toteudu. (Oracle 2009d, 4-7.)

4.3.4 Anonyymi lohko

Anonyymi lohko noudattaa ainoastaan aiemmin kuvassa 18 esiteltyä lohkon perusrakennetta, eikä sillä ole sen yksityiskohtaisempaa syntaksia, kuten seuraavissa kappaleissa olevilla proseduureilla ja funktioilla. Se on nimetön lohko, jota ei tallenneta kuten proseduureja tai funktioita. Siitä voidaan kutsua funktioita ja proseduureja, mutta sitä ei voi kutsua toisesta ohjelmasta, vaan se pitäisi ajaa sellaisenaan esimerkiksi proseduurin sisällä. Kuvassa 20 on esitetty yksinkertainen anonyymi lohko. (Oracle 2009d, 1-5.)

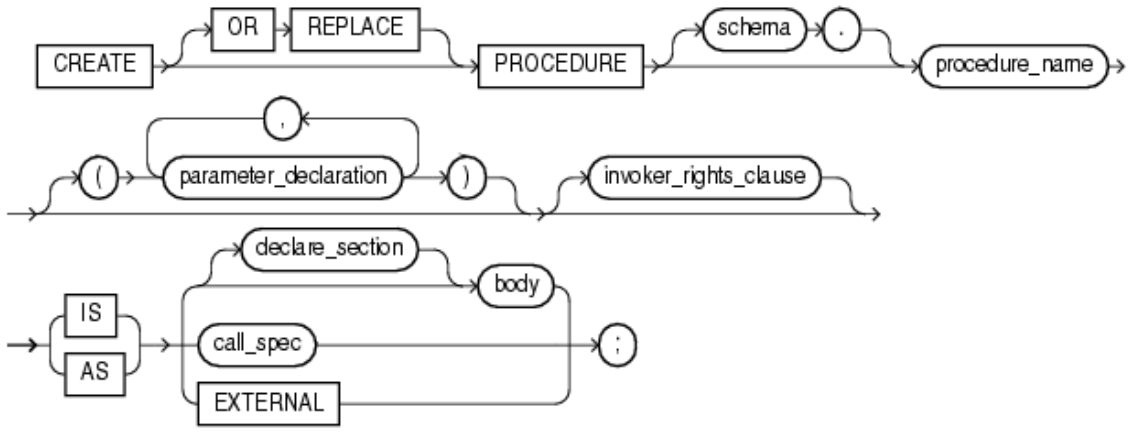
```
/* Hello World PL/SQL-kielellä */
BEGIN
    DBMS_OUTPUT.PUT_LINE('Hello world!');
END;

-- Ja sama käytettäessä muuttujaa text:

DECLARE
    text VARCHAR2(20);
BEGIN
    text := 'Hello world!';
    DBMS_OUTPUT.PUT_LINE(text);
END;
```

Kuva 20. Hello world -sovellus PL/SQL-kielellä

Kuvassa 20 olevassa anonyymissa lohkossa käytetään Oraclen omaa *dbms_output*-pakettia ja sen *put_line*-proseduuria, jolla voidaan tulostaa ulostulotietoa (*output*) tietokannasta. Ennen tiedon tulostamista täytyy kuitenkin huomioida, että ulostulotiedon tulostus on määritetty päälle *set serveroutput on* -komennolla. (Oracle 2009d, 1-6, 10-17)

4.3.5 Proseduuri

Proseduurit ovat aliohjelmia, jotka suorittavat niille määrätyt toiminnallisuudet. Ne ovat yleensä pieniä sovelluksen osia tai muita toimintoja, jotka manipuloivat tietokannan tietoja. Usein ne tallennetaan tietokantaan, ja siksi niitä voidaan kutsua myös nimellä talletetut proseduurit. Tallentamattomana se olisi anonyymi lohko. Proseduuri noudattaa lohkon perusrakennetta, mutta sillä on hieman yksityiskohtaisempi syntaksi (kuva 21). (Mielikäinen 2007a.)



Kuva 21. Proseduurin syntaksi (Oracle 2009d, 14-50)

Proseduuri tallentuu tietokantaan *create*-määreellä. *Replace*-määreen kanssa käytettynä aiempi samanniminen tietokannassa oleva proseduuri korvataan uudella. Proseduuria voidaan kutsua sovelluksesta, funktiosta, anonyymista lohkosta tai toisesta proseduurista, jolloin se suorittaa sille määritetyt toiminnallisuudet. Proseduurille voidaan määrittää myös parametreja, jotka annetaan sitä kutsuttaessa. Parametrille täytyy määrittää myös sen tietotyyppi. (Oracle 2009d, 13-105, 14-50.)

Procedure-määrityksen jälkeen esitellään proseduurissa käytettävät muuttujat ja kursorit. *Begin*-määreen jälkeen määritetään proseduurin toiminnallisuus ja *exception*-määreen jälkeen käsitellään mahdolliset poikkeustilanteet. Kuvassa 22 on esimerkki proseduurista, joka päivittää tauluja kutsuttaessa annettujen parametrien mukaisesti. (Oracle 2009d, 13-105, 14-50.)

```
CREATE OR REPLACE PROCEDURE new_salary(
    p_employee VARCHAR2, p_new_sal NUMBER) -- Proseduurin parametrit
AS
BEGIN
    UPDATE
        employee
    SET
        salary = p_new_sal
    WHERE
        emp_name = p_employee;
END;
```

Kuva 22. Esimerkki proseduurista

Kuvassa 23 anonyymi lohkossa kutsutaan *new_salary*-proseduuria kahdella parametrilla, jotka lähetetään proseduurille. Kutsuttaessa proseduuri suorittaa sen toiminnallisuudet, eli päivittää *employee*-taulun kenttää *salary*, jossa *emp_name*-kentän arvo on Erkki Esimerkki (1. parametri) arvoksi 3800 (2. parametri).

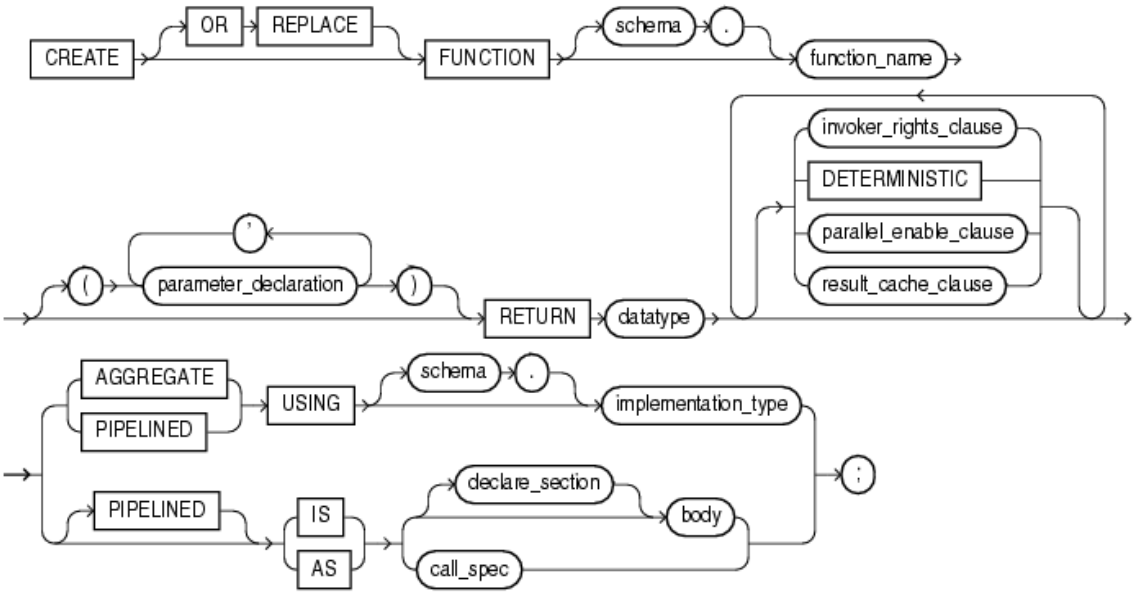
```
BEGIN
    new_salary('Erkki Esimerkki', 3800);
END;
```

Kuva 23. Proseduurin kutsuminen

4.3.6 Funktio

Funktiot ovat aliohjelmia, jotka on tarkoitettu laskemaan arvoja. Se on muulta raken-

teeltaan ja toiminnallisuudeltaan samanlainen kuin proseduuri, mutta funktiolla on aina *return*-lause, joka palauttaa arvon. Kantaan tallennettavan funktion syntaksi on esitetty kuvassa 24. (Oracle 2009d, 13-78.)



Kuva 24. Funktion syntaksi (Oracle 2009d 14-33)

Usein myös funktiot tallennetaan proseduurien tavoin tietokantaan *create or replace* -määreillä. Mikäli funktiota ei tallenneta tietokantaan, se on anonyymi lohko. Funktion määritys alkaa *function*-määreellä ja päättyy *return*-lauseeseen, joka kertoo, minkä tyyppistä tietoa funktio palauttaa. *Function*-määreen jälkeen esitellään muuttujat, kursorit ja poikkeukset. *Begin*-määreen jälkeen määritetään funktion toiminnallisuus, jossa myös palautetaan haluttu arvo. *Exception*-määreen jälkeen määritetään poikkeustilanteiden käsittely, kuten proseduureissa. (Oracle 2009d, 13-78, 14-33.)

Funktiota käytetään tietyn kokonaisuuden osana, eikä sitä voi kutsua lohkossa samalla tavalla, kuin proseduuria. Usein funktioita käytetään kontrollirakenteissa, kuten *if-else*-rakenteissa, joissa tutkitaan funktion palauttamaa arvoa, mutta sitä voidaan käyttää myös esimerkiksi *select*-, *insert*-, *update*-, *delete*- ja *where*-lauseissa. Kuvassa 25 on esimerkki funktiosta, jota kutsutaan kuvassa 26 *select*-lauseessa. (Oracle 2009d, 13-78, 14-33.)

```
/* Funktio laskee kaikkien tietyn läänin kaupunkien asukkaat */
CREATE OR REPLACE FUNCTION count_population(
       p_province VARCHAR2) RETURN NUMBER IS -- Lääni-parametri annetaan kutsuttaes-
sa
    province_population NUMBER;              -- Esitellään numerotyyppinen muuttuja
BEGIN
    SELECT
        sum(population)
            INTO province_population -- As.määrä laitetaan muuttujaan INTO-määreellä
    FROM
        city
    WHERE
        province name = p province;  -- Parametrilla rajataan kaupungit
    RETURN province_population;      -- Palautetaan asukasmäärä
END;
```

Kuva 25. Esimerkki funktiosta

```
SELECT
    count_population('Pirkanmaa') -- Funktio select-lauseessa
FROM
    dual;
```

Kuva 26. Funktion kutsuminen *select*-lauseessa

Kuvassa 26 oleva *dual*-taulu on Oraclen luoma aputaulu, jossa on ainoastaan yksi kenttä nimeltä *dummy* ja yksi rivi, jossa arvona *x*. Kaikilla tietokannan käyttäjillä on sen käyttöoikeus ja sitä käytetään usein *pseudokenttien*, kuten aikaleiman (*sysdate)* hakemiseen, joten sitä voi hyvin käyttää myös tässä esimerkissä sisällyttämään funktio *select*-lauseeseen. (Oracle 2009a, 9-15.)

4.3.7 Kursori

PL/SQL-kielessä kursori (*cursor*) on muuttujatyyppi, jolla pystyy hallitsemaan SQL-kyselyn tuloksia yksitellen. Kursori tarkoittaa SQL-kyselyä, joka käy kyselyn tuottamat rivit läpi ja osoittaa jokaiseen kyselyn tuottamaan riviin vuorollaan, jolloin niitä voi käsitellä yksitellen PL/SQL-koodissa. Kursoreita on olemassa kahdentyyppisiä: implisiittinen (*implicit*) ja eksplisiittinen (*explicit*) kursori. (Oracle 2009d, 6-6 - 6-19.)

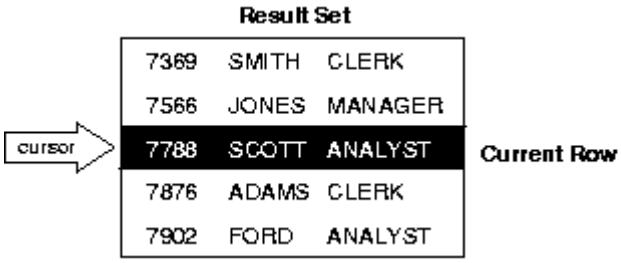
Implisiittinen kursori

Implisiittinen kursori tarkoittaa kursoria, jonka *select*-kysely suoritetaan. PL/SQL avaa kursorin kyselyn suorittamisen ajaksi ja sulkee sen, eikä se vielä tällä osoita mihinkään riviin. Siinä kyselyn tuloksia ei voi hallita, mutta sen attribuutteja voidaan käyttää niin kauan kunnes toinen *select*- tai *DML*-komento suoritetaan. Kursorin attribuutit ovat me-

tatietoja suoritetusta komennosta, esimerkiksi tuottiko ajettu *select*-kysely yhtään tulosta (*SQL%FOUND*). (Oracle 2009d, 6-6 - 6-19.)

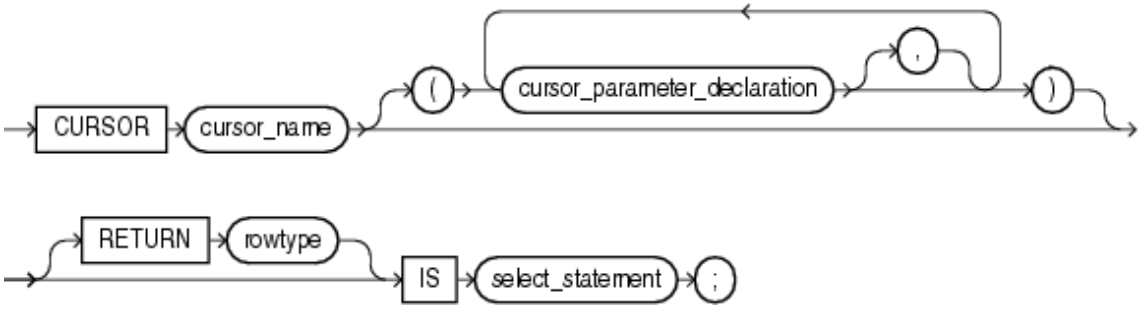
Eksplisiittinen kursori

Mikäli kursoria halutaan hallita koodissa ja käsitellä sen tuottamia rivejä yksitellen, täytyy kursori esitellä eksplisiittisesti. Eksplisiittistä kursoria pystyy hallitsemaan rivi kerrallaan esimerkiksi *for*-loopissa. Kaikki kursorit, jotka tuottavat yhden tai useamman rivin (yleensä useamman), ja joita käsitellään yksitellen koodissa, ovat eksplisiittisiä (kuva 27). (Oracle 2009d, 6-6 - 6-19.)



Kuva 27. Kursori osoittaa kyselyn tuottamaan riviin yksitellen (Oracle 2006c)

Kursori esitellään ja nimetään lohkon *declare*-osassa, ja sitä käsitellään lohkon toiminnallisuusosassa eli *begin*-määreen jälkeen. Kursoreissa voidaan käyttää myös parametreja, jolloin kursorin kyselyssä voidaan käyttää parametrin arvoja, jotka annetaan sitä kutsuttaessa. Kursorin voi myös määrittää palauttamaan arvon. Kursorin syntaksi on esitetty kuvassa 28. (Mielikäinen 2007c.)



Kuva 28. Kursorin syntaksi (Oracle 2009d, 13-56)

Kursoria voidaan käyttää erilaisilla tavoilla, kuten esimerkiksi avaamalla kursorin ensin *open*-määrellä ja kiinnittämällä rivi kerrallaan kursorin hakutulokset muuttujiin *fetch*-määreellä *loop*-silmukan sisällä. Silmukassa voidaan käyttää kursorin attribuuttia

%NOTFOUND tutkimaan, onko kursorissa vielä rivejä jäljellä, kuten kuvan 29 esimerkissä on tehty. Kursoria voidaan käyttää myös *for*-silmukassa, kuten myöhemmin esille tulevissa liitteen konversioskripteissä tehdään. (Oracle 2009d, 6-6 - 6-19.)

```
DECLARE
    v_city_name city.name%TYPE;
    v_city_pop city.population%TYPE;
    CURSOR c_populations IS
        SELECT ci.name, ci.population
        FROM city ci;
BEGIN
    OPEN c_populations;
    LOOP
        FETCH c_populations INTO v_city_name, v_city_pop;
        EXIT WHEN c_populations%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(v_city_name||': '||v_city_pop||' asukasta');
    END LOOP;
    CLOSE c_populations;
END;

/* Tulos muodossa:
     Tampere: 211350 asukasta
     Ylöjärvi: 30179 asukasta
     Nokia: 31362 asukasta
     Pirkkala: 16512 asukasta
     ...                      */
```

Kuva 29. Yksi esimerkki kursorin käytöstä

5 Konvertoitava tietokanta

Tässä luvussa esitettävän konversion tarkoituksena on havainnollistaa keinoja konversion tekemiseen ja tarkastella PL/SQL-kielen tehokkuutta konversiossa. Konversiossa käytetään tietokannan osalta täysin luotua – kuvitteellista tilannetta, jossa PL/SQL-kielen tehokkuus tulee hyvin esille. Konvertoitavan tietokannan rakenne on pyritty tekemään siten, että se noudattaisi mahdollisimman paljon tuotantokäytössä olevien tietokantojen toistuvia rakenteita.

Tässä luvussa kuvataan kuvitteellisen terveydenhuollon tietojärjestelmään liittyvää tietokantaa, jossa tietokannan rakennetta täytyy muuttaa siten, että se vastaisi järjestelmän tarpeita tulevassa pandemian massarokotustilanteessa. Massarokotuksiin osallistuvat kaikki terveydenhuollon laitokset ja tietokantaan halutaan oma tallennuspaikka tähän tarpeeseen.

Tietokannassa ei alun perin ole tietoa rokotuksiin liittyvistä asioista, joten kaikki sairaalat ja terveyskeskukset halutaan lisätä johonkin rokotusalueeseen. Lisäksi rokotusalueelle halutaan yksi koko alueesta vastaava sairaala. Rokotusalueen keskeisimmät sairaalat tulevat olemaan yliopistollisia keskussairaaloita.

Rokotusalueiden halutaan koostuvan seuraavista vastuualueista:

- Helsingin yliopistollisen keskussairaala rokotusalue (HYKS Rokotusalue)
- Kuopion yliopistollisen keskussairaalan rokotusalue (KYS Rokotusalue)
- Oulun yliopistollisen keskussairaalan rokotusalue (OYS Rokotusalue)
- Tampereen yliopistollisen keskussairaalan rokotusalue (TAYS Rokotusalue)
- Turun yliopistollisen keskussairaalan rokotusalue (TYKS Rokotusalue).

Kaikki terveydenhuollon laitokset jaetaan näihin alueisiin, riippuen siitä, missä sairaanhoitopiirissä laitos sijaitsee. Tietokantaan tulee siis lisätä tieto uusista rokotusalueista määrittää niille kyseisestä alueesta vastaava yliopistollinen keskussairaala. Lisäksi tietokantaan halutaan saada tieto jokaisen sairaalan ja terveyskeskuksen kuulumisesta tiettyyn rokotusalueeseen, jotka käytännössä toimivat rokotuspaikkoina.

5.1 Vanha taulurakenne

Tässä alaluvussa esitetään olemassa oleva tietokanta, jota lähdetään konvertoimaan uuteen rakenteeseen. Tietokannassa on ennestään tieto sairaanhoitopiireistä, sairaaloista, terveyskeskuksista ja sairaanhoitopiirien keskussairaaloista. Vanha taulurakenne sisältää *hospital*-, *clinic*-, *healthcare_area*- ja *area_central_hospital*-taulut. Taulukoissa 1 - 4 esitetään vanhan taulurakenteen taulut ja niiden kentät.

5.1.1 Taulut

Healthcare_area-taulu

Healthcare_area-taulu sisältää sairaanhoitopiirit, joita on Suomessa 20 kappaletta. Taulu sisältää pääavaimen, sairaanhoitopiiri nimen ja rivin luonnin tunnistetiedot eli aikaleiman ja rivin luoneen toiminnon (taulukko1). Rivin aikaleimassa käytetään Oraclen omaa *sysdate*-pseudokenttää, ja rivin luoneessa toiminnossa käytetään tekstiä *Opinnäytetyö*. Nämä tunnistetiedot ovat tietokannan jokaisessa taulussa, sillä tunnistekenttien avulla on helppo myöhemmin nähdä, koska ja mikä toiminto kyseisen rivin on luonut. Tunnistekenttien käyttö tauluissa on hyvä tapa, sillä isoissa tietokannoissa niistä voi olla korvaamatonta apua vanhojen tai epäilyttävien rivien tarkoituksen selvittämisessä.

Taulukko 1: Healthcare_area-taulun kentät

Kenttä	Kentän selitys	Kentän tyyppi	Not null	Viittaus
healthcare_area_id	Pääavain	number(12)	X	
name	Sairaanhoitopiirin nimi	varchar2(255)		
row_timestamp	Rivin aikaleima	date	X	
row_creator	Rivin luonut toiminto	varchar2(255)	X	

Hospital-taulu

Hospital-taulussa ovat sairaalat, keskussairaalat ja yliopistolliset keskussairaalat. Taulu viittaa sairaanhoitopiiriin *healthcare_area_id*-kentällä, joka on *not null* -tyyppien, joten kaikkien sairaaloiden täytyy kuulua johonkin sairaanhoitopiiriin. Taulussa on lisäksi pääavain *hospital_id*- sekä *name*-, *personnel*- ja tunnistetietokentät. *Name*-kenttä on sairaalan nimi ja *personnel*-kentässä on sairaalan henkilöstön lukumäärä.

Talukko 2: Hospital-taulun kentät

Kenttä	Kentän selitys	Kentän tyyppi	Not null	Viittaus
hospital_id	Pääavain	number(12)	X	
healthcare_area_id	Viiteavain	number(12)	X	healthcare_area. healthcare_area_id
name	Sairaalan nimi	varchar2(255)		
personnel	Henkilöstön lukumäärä	number		
row_timestamp	Rivin aikaleima	date	X	
row_creator	Rivin luonut toiminto	varchar2(255)	X	

Area_central_hospital-taulu

Area_central_hospital-taulu on *hospital*- ja *healthcare_area* -taulujen välinen yhteystaulu, joka määrittää sairaanhoitopiirin keskussairaalan (taulukko 3 ja kuva 30). Taulussa on yksi rivi jokaista sairaanhoitopiiriä kohti. Rivi kertoo *healthcare_area_id*-kentällä sairaanhoitopiirin ja *hospital_id*-kentällä kyseisen sairaanhoitopiirin keskussairaalan. Nämä kentät ovat vierasavaimia, mutta yhdessä ne toimivat myös pääavaimena. Taulussa on siis kaksiosainen pääavain, joka yksilöi tietueen.

Taulukko 3: Area_central_hospital-taulun kentät

Kenttä	Kentän selitys	Kentän tyyppi	Not null	Viittaus
healthcare_area_id	Pääavain, Viiteavain	number(12)	X	healthcare_area. healthcare_area_id
hospital_id	Pääavain, Viiteavain	number(12)	X	hospital.hospital_id
row_timestamp	Rivin aikaleima	date	X	
row_creator	Rivin luonut toiminto	varchar2(255)	X	

Clinic-taulu

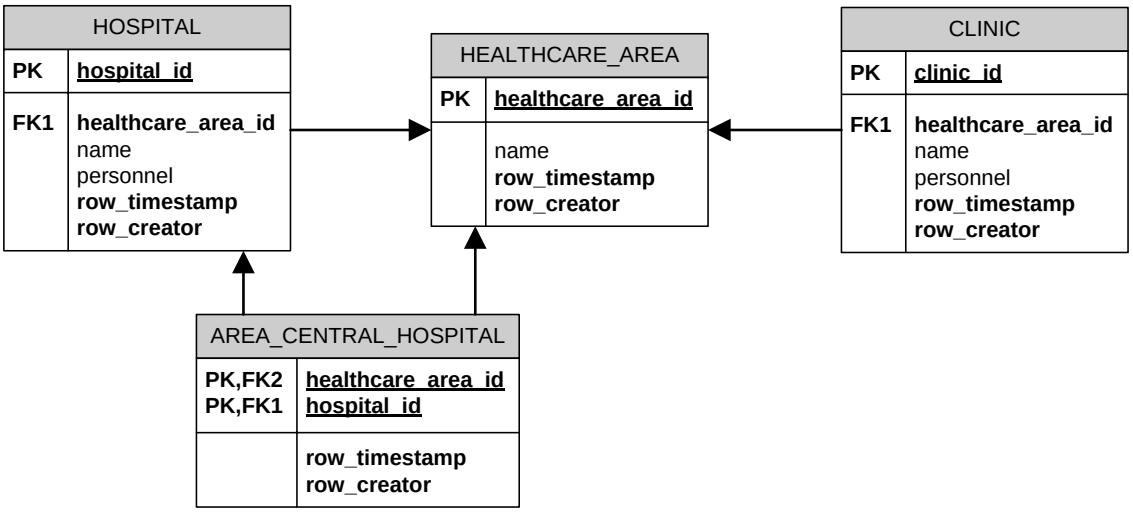
Clinic-taulussa on tieto terveyskeskuksista. Myös terveyskeskukset kuuluvat sairaaloiden tapaan aina johonkin sairaanhoitopiirin. Taulu viittaa *healthcare_area*-tauluun viiteavaimella *healthcare_area_id*. Lisäksi taulussa on *name*-kenttä terveyskeskuksen nimelle, *personnel*-kenttä henkilöstön lukumäärälle ja tunnistetiedot, kuten taulukossa 4 on esitetty.

Taulukko 4: Clinic-taulun kentät

Kenttä	Kentän selitys	Kentän tyyppi	Not null	Viittaus
clinic_id	Pääavain	number(12)	X	
healthcare_area_id	Viiteavain	number(12)	X	healthcare_area. healthcare_area_id
name	Terveyskeskuksen nimi	varchar2(255)		
personnel	Henkilöstön lukumäärä	number		
row_timestamp	Rivin aikaleima	date	X	
row_creator	Rivin luonut toiminto	varchar2(255)	X	

5.1.2 Taulujen yhteydet

Kuvan 30 relaatiokaaviossa näkyy vanhan taulurakenteen taulujen yhteydet toisiinsa. Taulusta lähtevä nuoli tarkoittaa, että taulussa on viiteavain nuolen osoittamaan tauluun. Viiteavain on aina samanniminen, kuin nuolen osoittaman taulun pääavain. Esimerkiksi *clinic*- ja *hospital*-taulujen *healthcare_area_id*-kentät viittaavat *healthcare_area*-taulun samannimiseen kenttään.



Kuva 30: Vanhan taulurakenteen relaatiokaavio

5.2 Uusi taulurakenne ja määrittely

Area_central_hospital-taulusta halutaan päästä eroon ja tilalle luodaan *central_hospital*-taulu, johon sairaanhoitopiirien keskussairaalat siirretään *hospital*-taulusta. Näihin kuu-

luvat yliopistolliset keskussairaalat ja muut keskussairaalat. Keskussairaalat poistetaan *hospital*-taulusta, kun ne on saatu siirrettyä *central_hospital*-tauluun. Lisäksi *healthcare_area*-taulua muutetaan siten, että se viittaa suoraan *central_hospital*-tauluun. Näin sairaanhoitopiiri viittaa suoraan kyseisen sairaanhoitopiiri keskussairaalaan, ilman välitaulua *area_central_hospital*.

Taulukoissa 5, 7 ja 8 esitetään uusien taulujen kenttien määritykset ja taulukossa 9 muutettavan *health_care_area*-taulun kentät. *Hospital*-taulun kenttiä ei tarvitse muuttaa, sillä sieltä ainoastaan poistetaan tietueita *delete*-komennolla. *Hospital*-taulun poistettavien rivien tietoja tarvitaan *central_hospital*-taulun konversiossa, joten tämä täytyy tehdä ennen rivien poistamista *hospital*-taulusta.

5.2.1 Poistettavat taulut

Area_central_hospital

Sairaalan ja sairaanhoitopiirin välinen *area_central_hospital*-taulu poistetaan rakenteesta. Tätä ei kuitenkaan voida tehdä ennen kuin *healthcare_area*-tauluun on luotu suorat viittaukset *central_hospital*-tauluun. *Health_care_area*-taulun konversiossa siis tarvitaan tietoa siitä, mikä sairaala toimii minkäkin sairaanhoitopiirin keskussairaalana.

5.2.2 Uudet taulut

Vaccine_area

Vaccine_area-tauluun tallennetaan rokotusalueet. Rokotusalueella on koko alueesta vastaava yliopistollinen keskussairaala, joten taulu tulee viittaamaan *central_hospital*-tauluun *central_hospital_id*-kentällä. Taulussa on lisäksi rokotusalueen nimi ja tunnistetiedot, kuten taulukossa 5 on esitetty.

Taulukko 5: Vaccine_area-taulun kentät

Kenttä	Kentän selitys	Kentän tyyppi	Not null	Viittaus
vaccine_area_id	Pääavain	number(12)	X	
central_hospital_id	Viiteavain	number(12)	X	central_hospital. central_hospital_id
name	Rokotusalueen nimi	varchar2(255)		
row_timestamp	Rivin aikaleima	date	X	
row_creator	Rivin luonut toiminto	varchar2(255)	X	

Taulukossa 6 on esitetty sairaanhoitopiirit, joista rokotusalueet tulevat koostumaan. Rokotusalueita on 5 kappaletta, eli jokaisella yliopistollisella keskussairaalalla on oma rokotusalueensa, josta se vastaa. Kaikki sairaalat ja terveyskeskukset tulevat kuulumaan tietokannassa johonkin rokotusalueeseen. Esimerkiksi Hatanpään terveysasema kuuluu Pirkanmaan sairaanhoitopiiriin, jolloin se tulee kuulumaan Taysin rokotusalueeseen.

Taulukko 6: Rokotusalueet

HYKS Rokotusalue
Helsingin ja Uudenmaan sairaanhoitopiiri
Etelä-Karjalan sairaanhoitopiiri
Kymenlaakson sairaanhoitopiiri
KYS Rokotusalue
Pohjois-Savon sairaanhoitopiiri
Etelä-Savon sairaanhoitopiiri
Itä-Savon sairaanhoitopiiri
Keski-Suomen sairaanhoitopiiri
Pohjois-Karjalan sairaanhoitopiiri
OYS Rokotusalue
Pohjois-Pohjanmaan sairaanhoitopiiri
Länsi-Pohjan sairaanhoitopiiri
Lapin sairaanhoitopiiri
Kainuun sairaanhoitopiiri
Keski-Pohjanmaan sairaanhoitopiiri
TAYS Rokotusalue
Pirkanmaan sairaanhoitopiiri
Etelä-Pohjanmaan sairaanhoitopiiri
Kanta-Hämeen sairaanhoitopiiri
Päijät-Hämeen sairaanhoitopiiri
Vaasan sairaanhoitopiiri
TYKS Rokotusalue
Varsinais-Suomen sairaanhoitopiiri
Satakunnan sairaanhoitopiiri

Vaccine_provider

Vaccine_provider-tauluun lisätään jokaista sairaalaa ja terveyskeskusta kohden yksi rivi. Taulu viittaa *hospital_id*- tai *clinic_id*-kentällä joko sairaalaan tai terveyskeskukseen ja *vaccine_area_id*-kentällä siihen rokotusalueeseen, johon sairaala tai terveyskeskus kuuluu. *Hospital_id*- ja *clinic_id*-kentissä ei siis voi olla arvoa yhtä aikaa. Rivi viittaa joko sairaalaan tai terveyskeskukseen, riippuen siitä, minkä sairaalan tai terveyskeskuksen kuulumista tiettyyn alueeseen rivi kuvaa. Siksi *hospital_id*- ja *clinic_id*-kentät eivät voi olla *not null* -tyyppisiä. *Vaccine_provider*-taulun kentät on esitetty taulukossa 7.

Taulukko 7: Vaccine_provider-taulun kentät

Kenttä	Kentän selitys	Kentän tyyppi	Not null	Viittaus
vaccine_provider_id	Pääavain	number(12)	X	
vaccine_area_id	Viiteavain	number(12)	X	vaccine_area. vaccine_area_id
hospital_id	Viiteavain	number(12)		hospital.hospital_id
clinic_id	Viiteavain	number(12)		clinic.clinic_id
row_timestamp	Rivin aikaleima	date	X	
row_creator	Rivin luonut toiminto	varchar2(255)	X	

Central_hospital

Central_hospital-taulu luodaan sairaanhoitopiirien keskussairaaloiden tallennuspaikoiksi. Taulu tulee sisältämään joko keskussairaaloita tai yliopistollisia keskussairaaloita. Näitä on yhteensä 20 kappaletta, kuten itse sairaanhoitopiirejäkin. Esimerkiksi Tampereen yliopistollinen keskussairaala on Pirkanmaan sairaanhoitopiirin keskussairaala. Taulun kentät on esitetty taulukossa 8.

Taulukko 8: Central_hospital-taulun kentät

Kenttä	Kentän selitys	Kentän tyyppi	Not null	Viittaus
central_hospital_id	Pääavain	number(12)	X	
name	Keskus- tai yliopistollisen sairaalan nimi	varchar(255)		
personnel	Henkilöstön lukumäärä	number		
row_timestamp	Rivin aikaleima	date	X	
row_creator	Rivin luonut toiminto	varchar2(255)	X	

5.2.3 Muutettavat taulut

Healthcare_area

Tauluun lisätään uusi kenttä *central_hospital_id*, joka viittaa *central_hospital*-tauluun. Näin sairaanhoitopiiri saadaan viittamaan suoraan sen keskussairaalaan ilman minkäänlaisia välitauluja. Taulukossa 9 on esitetty *health_care_area*-taulun uusi rakenne.

Taulukko 9: Healthcare_area-taulun kentät

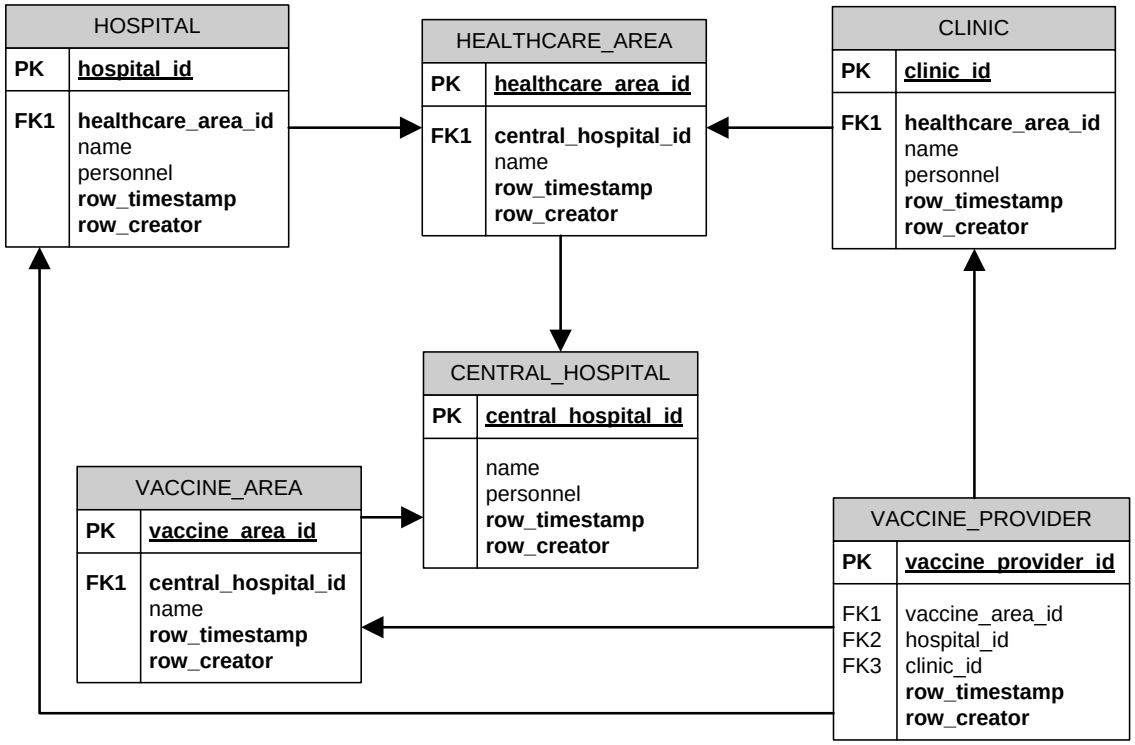
Kenttä	Kentän selitys	Kentän tyyppi	Not null	Viittaus
healthcare_area_id	Pääavain	number(12)	X	
central_hospital_id	Viiteavain	number(12)	X	central_hospital. central_hospital_id
name	Sairaanhoitopiirin nimi	varchar2(255)		
row_timestamp	Rivin aikaleima	date	X	
row_creator	Rivin luonut toiminto	varchar2(255)	X	

Hospital

Hospital-taulusta poistetaan sairaanhoitopiirien keskussairaaloita kuvaavat tietueet, jotka on sitä ennen lisätty uuteen tallennuspaikkaansa, central_*hospital*-tauluun. Poistettavia rivejä on yhteensä 20 kappaletta. Ennen rivien poistamista ne täytyy konvertoida *central_hospital*-tauluun.

5.2.4 Taulujen uudet yhteydet

Kuvan 30 relaatiokaaviossa on määritelty taulujen yhteydet uudessa rakenteessa. Relaatiokaaviosta voi huomata, että *area_central_hospital*-taulu on poistettu ja tilalle on luotu *central_hospital*-taulu. Lisäksi kuvaan on määritetty uudet rokotuksiin liittyvät taulut *vaccine_area* ja *vaccine_provider* sekä niiden yhteydet muihin tauluihin.



Kuva 31: Uuden taulurakenteen relaatiokaavio

5.3 Konversioskriptit

Tässä alaluvussa kerrotaan yksityiskohtaisesti tehtävät toimenpiteet konversion aikaansaamiseksi ja viitataan liitteessä 1 oleviin konversioskripteihin, joilla konversion käytännön osuus toteutetaan. Konversioskriptejä on kymmenen kappaletta ja ne ovat liitteessä 1 numeroituna yhdestä kymmeneen siinä järjestyksessä, jossa ne tulee ajaa.

Konversio aloitetaan luomalla uudet taulut. Tämän jälkeen tehdään määrittelyn mukaisesti datakonversio, jossa uusiin tauluihin lisätään tietoa ja olemassa olevien taulujen tietoja muokataan. Lopuksi poistetaan turhat taulut ja rivit.

5.3.1 Taulujen luonti

Konversioskripteissä 1-3 luodaan uudet taulut *central_hospital*, *vaccine_area* ja *vaccine_provider*. Kaikki pääavaimet määritetään *number*-tyyppiseksi ja 12 merkin pituisiksi. Tekstityyppiset kentät, kuten *name* ja *row_creator* määritetään *varchar2*-tyyppisiksi ja 255 merkin pituisiksi. *Row_timestamp*-kenttiin määritetään *date*-tietotyyppi, joka

sopii aikaleiman (*sysdate*) tietotyypiksi. Taulujen luontilauseet sisältävät myös viiteavaimien määrityksen.

5.3.2 Taulujen muokkaus

Konversioskriptissä 4 *healthcare_area*-tauluun luodaan uusi kenttä *central_hospital_id* ja siihen määritetään viiteavain viittaamaan *central_hospital*-taulun samannimiseen kenttään. Näin saadaan sairaanhoitopiirit viittamaan suoraan sairaanhoitopiirien keskussairaaloihin, ilman välitaulua. *Central_hospital_id*-kentän data lisätään datakonversiossa sen jälkeen, kun keskussairaalat on saatu konvertoitua niiden uuteen tallennuspaikkaansa – *central_hospital*-tauluun, jolloin näihin riveihin voidaan viitata.

5.3.3 Datakonversio

Keskussairaalat

Konversioskriptissä 5 konvertoidaan äsken luotuun *central_hospital*-tauluun sairaanhoitopiirien keskussairaalat. Keskussairaalat ovat tällä hetkellä *hospital*-taulussa, jossa muutkin sairaalat ovat. Tässä käytetään apuna *area_centra_hospital*-taulua, jossa ovat viittaukset alueiden keskussairaaloihin.

Konversioskripti aloitetaan luomalla kursori nimeltä *c_central_hospitals*, joka hakee keskussairaaloiden pääavaimet, nimet ja henkilöstön lukumäärän. Kursorin kysely rajataan niihin *hospital*-taulun riveihin, joiden pääavaimet löytyvät *area_central_hospital*-taulusta. Näin kursorin kysely tuottaa tulokseksi ainoastaan nämä 20 sairaalaa, jotka toimivat sairaanhoitopiirien keskussairaaloina.

Konversioskriptin toiminnallisuusosassa ”loopataan” silmukkarakenteessa jokainen kursorin tuottama rivi ja lisätään *central_hospital*-tauluun rivi jokaiselle keskussairaalalle. Pääavain, nimi ja henkilöstön lukumäärä otetaan *insert*-lauseeseen kursorin hakutuloksista, joten ne pysyvät samoina kuin mitä ne olivat *hospital*-taulussakin. Tunnistetietoihin lisätään aikaleimatieto (*sysdate*) ja tietueen luonut toiminto. *V_count*-muuttuja laskee, montako kertaa silmukka pyörähtää, ja kun kursori ei anna enää hakutuloksia,

siirrytään pois silmukasta, tulostetaan lisättyjen rivien lukumäärä ja hyväksytään transaktio. Konversioskriptin lopussa on poikkeuksien käsittely, mikäli skripti antaisi virheilmoituksen.

Viittaukset keskussairaaloihin

Kun rivit on luotu *central_hospital*-tauluun, voidaan lisätä viittaukset näihin *healthcare_area*-taulusta. Konversioskriptissä 6 luodaan *c_central_hospital_ids*-kursori, joka hakee *hospital_id*- ja *healthcare_area_id*-kenttien tiedot *area_central_hospital*-taulusta. Näin saamme tiedon siitä, mikä sairaala on minkäkin sairaanhoitopiirin keskussairaala.

Toiminnallisuusosassa ”loopataan” kursoria ja päivitetään *healthcare_area*-taulun uuteen *central_hospital_id*-kenttään viittaus keskussairaalaan, jossa viittaus on kyseiseen sairaanhoitopiirin. Samalla lisätään tunnistetiedot ja lopuksi tulostetaan, montako riviä päivitettiin ja hyväksytään transaktio.

Rokotusalueet

Konversioskriptissä 7 konvertoidaan rokotusalueet, jotka on määritelty luvussa 5. Rokotusalueita on 5 kappaletta, eli jokaisella yliopistollisella keskussairaalalla on oma rokotusalueensa, josta se vastaa.

Konversioskriptin *declare*-osassa määritetään *c_central_hospital*-kursori, joka hakee kaikkien yliopistollisten keskussairaaloiden pääavaimet ja nimet, sekä esitellään muuttujat *v_count* ja *v_name*. *V_count*-muuttujalla ainoastaan lasketaan lisättyjen rivien lukumäärän, kuten aiemmissa konversioskripteissä. *V_name*-muuttujaa puolestaan käytetään toiminnallisuusosan *case*-rakenteessa määrittämään oikean rokotusalueen nimen, riippuen mihin yliopistolliseen keskussairaalaan kursori viittaa. Tällöin muuttujan nimi on se rokotusalue, jonka keskeisempänä sairaalana kyseinen yliopistollinen sairaala on.

Toiminnallisuusosan *insert*-komennossa luodaan rivi rokotusalueelle, jossa *hc.nextval*-komennolla pääavaimeksi annetaan *hc*-nimisen sekvenssin seuraava vapaa arvo. Sekvenssi voidaan ymmärtää jonkinlaisena vapaiden pääavaimien ”altaana”, josta *nextval*-komennolla saadaan seuraava vapaa yksilöllinen pääavain. Sitten *insert*-komennossa lisätään riville oikea viiteavain viittamaan yliopistolliseen keskussairaalaan ja *v_name*-muuttuja rokotusalueen nimeksi, riippuen siitä, mikä yliopistollinen keskussairaala on

kyseessä sekä tunnistetiedot. Kun kursori ei anna enää hakutuloksia, siirrytään silmukasta pois, tulostetaan lisättyjen rivien lukumäärä ja hyväksytään transaktio.

Sairaalat ja terveyskeskukset rokotusalueisiin

Sairaaloiden ja terveyskeskuksien lisääminen rokotusalueisiin toteutetaan lisäämällä *vaccine_provider*-tauluun jokaista sairaalaa ja terveyskeskusta kohden yksi rivi. Rivi viittaa rokotusalueeseen ja sairaalaan tai terveyskeskukseen. Konversioskriptissä 8 luodaan kursori, joka hakee sairaaloiden ja terveyskeskuksien pääavaimet ja niiden sairaanhoitopiirien nimet. Haku on toteutettu *union*-lauseella, eli sairaalat haetaan omassa kyselyssä ja terveyskeskukset omassa. Kumpaankin kyselyyn lisätään kirjain *H* tai *C* kertomaan, kummasta kyselystä on kyse – sairaaloista vai terveyskeskuksista. Tällä voidaan toiminnallisuusosassa erotella sairaalat ja terveyskeskukset *if-else*-rakenteella. Jos kyseessä on sairaala, lisätään *vaccine_provider*-taulun *hospital_id*-kenttään sairaalan pääavain, ja jos kyseessä on terveyskeskus, lisätään *vaccine_provider*-taulun *clinic_id*-kenttään terveyskeskuksen pääavain.

Toiminnallisuusosassa *case*-rakenteessa tutkitaan, mihin sairaanhoitopiiriin laitos kuuluu. Muuttujaa *v_name* käytetään apuna määrittämään sille oikea rokotusalue. Kun oikea rokotusalue on tiedossa, haetaan *execute immediate* -komennolla kyseisen rokotusalueen pääavain *v_vaccine_area_id*-muuttujaan. PL/SQL-kielessä *Execute immediate* -komento suorittaa SQL-komennon lohkon sisällä välittömästi, ja *into*-määreellä hakutulos sijoitetaan muuttujaan. Sitten luodaan rivi omalla *insert*-komennolla joko sairaalalle tai terveyskeskukselle riippuen siitä, onko kursorin hakutuloksessa tunnistuskirjain *H* vai *C*. Lopuksi tulostetaan luotujen rivien lukumäärän ja hyväksytään transaktio.

5.3.4 Taulujen ja rivien poistaminen

Lopuksi voidaan poistaa turha *area_central_hospital*-taulu ja tuhota keskussairaaloita kuvaavat rivit *hospital*-taulusta. Konversioskriptissä 9 tuhotaan *area_central_hospital*-taulu, joka täytyy tehdä ennen rivien poistamista *hospital*-taulusta, koska muuten viiteeheys rikkoutuisi, ja komento antaisi virheilmoituksen. Konversioskriptissä 10 poistetaan lopuksi keskussairaaloita kuvaavat rivit *hospital*-taulusta.

5.4 Konversio

Itse konversio on käytännössä vain skriptien ajamista tietokantaan ja lopputuloksen varmistaminen. Tähän pääseminen vaatii kuitenkin paljon esityötä, kuten tarkkaa määrittelyä ja säntillistä testaamista. Varsinkin isoissa ja kriittisissä järjestelmissä tietokannan konversion tulisi onnistua kerralla, eikä isoihin virheisiin ole varaa.

Ennen skriptien ajamista tietokantaan, täytyy jokainen yksittäinen konversioskripti testata, jotta sen lopputulos on varmasti oikea. Skriptejä voidaan testata tietokantaan jättämällä *commit*-komento antamatta ja tarkastelemalla skriptin tuottamia tuloksia, kun kyseessä on DML-komento. Toinen tapa on testata skriptejä sitä varten luotuihin aputauluihin ja varmistua siitä, että skripti tekevät halutut toimenpiteet. Kun voidaan olla varmoja, että konversioskriptit varmasti tekevät halutut toimenpiteet, voidaan hyväksyä transaktio *commit*-komennolla.

Yrityksissä voi olla myös käytössä tuotantotietokannan rinnalla jonkinlainen tuotantotietokantaa vastaava testi- tai kehitystietokanta, jossa kaikki muutokset testataan ennen tuotantoon siirtoa. Suurissa uudistus- ja kehityshankkeissa tämä on hyvä toimintatapa, koska tuotantotietokantaan tehdään näin ainoastaan varmasti toimivat muutokset, jotka on voitu testata kunnolla kehitystietokannassa. Tämä onkin suositeltavin tapa toteuttaa konversio.

Isoissa ja kriittisissä järjestelmissä testitietokanta on usein välttämätön. Eihän esimerkiksi rautateiden kulunvalvontajärjestelmää voi pitää suljettuna pitkiä aikoja, sillä sen avulla valvotaan, etteivät junat törmäile toisiinsa. Siksi määrittelyt ja testaukset tulisi tehdä niin kattavasti jo testitietokannassa, että tuotannossa olevaan tietokantaan skriptit pystytään ajamaan ja varmistamaan hetkessä.

6 Yhteenveto

Opinnäytetyön ensisijaisena tavoitteena oli kuvata Oracle-pohjaisen tietokannan konversioprosessia, jotta tuloksia voitaisiin käyttää hyödyksi kaikenlaisten Oracle-pohjaisten tietokantojen muuntamisessa ja kehittämisessä. Tähän tavoitteeseen päästiin, sillä opinnäytetyössä tuli esille monia erilaisia tapoja ja esimerkkejä tietokannan rakenteiden ja datan muuntamiseen. Lisäksi SQL- ja PL/SQL-kielien tärkeimmät ominaisuudet ja erilaiset käyttötavat esiteltiin esimerkein.

Opinnäytetyön konversiotilanteessa tavoite oli kuvata konversiossa käytettäviä keinoja siten, että niitä voitaisiin hyödyntää myös muihin tietokantoihin sekä kuvata konversioprosessin eri vaiheet. Myös tähän tavoitteeseen päästiin. Konvertoitavan tietokannan rakenne, määrittely ja konversio kuvattiin vaihe vaiheelta. Näitä tuloksia voidaan hyödyntää kaikenlaisiin tietokannan konversio- tai kehitystöihin.

Koska opinnäytetyö keskittyi ensisijaisesti Oracle-tietokantoihin, oli Oraclen PL/SQL-kielen käyttäminen konversion toteuttamisessa johdonmukainen valinta. Konversiossa onkin pyritty pitämään PL/SQL-kieli pääosassa ja myös hieman helpommat datakonversiot ovat tehty sen avulla.

SQL-kielen avulla voi luoda monimutkaisia tietokannan hallintaskriptejä, ja osa konversion toimenpiteistä olisi ollut tehtävissä myös siten. Esimerkiksi viiden rokotusalueen lisääminen tauluun olisi onnistunut hyvin helposti viidellä *insert*-lauseella, joten dynaamisen PL/SQL-lohkon kirjoittaminen siihen tehtävään oli ehkä ylimitoitettua. Kuitenkin on paljon tilanteita, jossa PL/SQL-kieli on SQL-kieleen verrattuna huomattavasti tehokkaampi. Tietokannan ollessa suurempi, voi yksittäisten *insert*-komentojen kirjoittaminen tai SQL komentojen pitkä putkittaminen käydä hyvin työlääksi. Tällöin proseduraalinen laajennus SQL-kieleen helpottaa suuresti tietokannan hallintaa.

Konversiotilanteessa selvisi sekin, että konversiotilanteessa PL/SQL-kielen käyttö voi olla tehokasta, mutta kaikkeen sitä ei välttämättä kannata käyttää. SQL-kieli on PL/SQL-kieleen verrattuna hieman helpommin ymmärrettävää, eikä sen kirjoittamiseen välttämättä vaadita keskittymistä samalla tavalla, kuin PL/SQL-kielessä. Jonkinlaisena yleistyksenä voisi pitää sitä, että SQL-kielellä voi ehtiä luomaan kymmeniä yksinkertai-

sia luonti-, muokkaus- ja poistolauseita ennen kuin PL/SQL-lohkon saa toimimaan halutulla tavalla. PL/SQL-kieltä voikin valehtelematta pitää täysipainoisena ohjelmointikielenä, jolla voi rakentaa kokonaisia järjestelmiä.

Jatkossa voisi olla mielenkiintoista tarkastella PL/SQL-kieltä kokonaisten tietojärjestelmien kehittämisessä, ja sitä, onko se enää nykypäivänä varteenotettava ratkaisu isojen tietojärjestelmien luontiin. Tässä opinnäytetyössä on tarkasteltu PL/SQL-kieltä pelkästään tietokannan hallinnassa, mutta myöhemmin näkökulmaksi voisi ottaa tietojärjestelmien kehittämisen PL/SQL-kielen avulla.

Lähteet

Hernandez, Michael J. 2000. Tietokannat - Suunnittelu käytännössä. Jyväskylä: Gummerus Kirjapaino Oy.

Hovi, Ari 1996. SQL-opas. Jyväskylä: Gummerus Kirjapaino Oy

Hovi, Ari 2000. SQL-ohjelmointi - Pro Training. Jyväskylä: Gummerus Kirjapaino Oy

Kettunen, Eero 2000. Johdatus PL/SQL-ohjelmointiin. [online] [viitattu 13.10.2009] http://www.lpt.fi/it/opetus/tietokannat/plsql.html

Mielikäinen, Maisa 2007a. Tietokantasovellukset. [online] [viitattu 27.3.2010] http://ta.ramk.fi/~maisa.mielikainen/504O04_Tietokantasovellukset/Talletetut_proseduurit.pdf

Mielikäinen, Maisa 2007b. Tietotyypit ja muuttujat. [online] [viitattu 27.3.2010] http://ta.ramk.fi/~maisa.mielikainen/A504O05_Tietokantasovellukset/Tietotyypeista_ja_muuttujista.pdf

Mielikäinen, Maisa 2007c. PL/SQL Kursorit. [online] [viitattu 27.3.2010] http://ta.ramk.fi/~maisa.mielikainen/504O04_Tietokantasovellukset/Luento3_Kursorit.pdf

Mureakuha 2006. Yleistä tietokannoista. [online] [viitattu 27.3.2010] http://wiki.mureakuha.com/wiki/Yleist%C3%A4_tietokannoista

Oracle 2006a. TopLink Developer’s Guide [online] [viitattu 27.3.2010] http://download.oracle.com/docs/cd/B32110_01/web.1013/b28218/relmapun.htm

Oracle 2006b. Application Server High Availability Guide [online] [viitattu 27.3.2010] http://download.oracle.com/docs/cd/B28196_01/core.1014/b28186/failoverinoracs.htm

Oracle 2006c. PL/SQL User's Guide and Reference [online] [viitattu 27.3.2010] http://download-west.oracle.com/docs/cd/B10501_01/appdev.920/a96624/01_oview.htm#740

Oracle 2009a. SQL Language Reference 11g Release 2. [online] [viitattu 27.3.2010] http://download.oracle.com/docs/cd/E11882_01/server.112/e10592.pdf

Oracle 2009b. Concepts 11g Release 2. [online] [viitattu 27.3.2010] http://download.oracle.com/docs/cd/E11882_01/server.112/e10713.pdf

Oracle 2009c. Administrator's Guide 11g Release 2. [online] [viitattu 27.3.2010] http://download.oracle.com/docs/cd/E11882_01/server.112/e10595.pdf

Oracle 2009d. PL/SQL Language Reference 11g Release 2. [online] [viitattu 27.3.2010] http://download.oracle.com/docs/cd/E11882_01/appdev.112/e10472.pdf

Tietokone 2010. Kiistelty Oracle–Sun-kauppa sai EU:n siunauksen. [online] [viitattu 27.3.2010] http://www.tietokone.fi/uutiset/kiistelty_oracle_sun_kauppa_sai_eu_n_siunauksen

Liitteet

Liite 1: Konversioskriptit

```
/*
  20.02.2010 / Juho Kangas: Konversioskripti 1.
  Luodaan uusi taulu central_hospital sairaanhoitopiirien keskussairaaloille.
*/
CREATE TABLE central_hospital(
  central hospital id      NUMBER(12)    NOT NULL PRIMARY KEY,
  name                     VARCHAR2(255),
  personnel                NUMBER,
  row timestamp            DATE          NOT NULL,
  row_creator              VARCHAR2(255) NOT NULL);


/*
  20.02.2010 / Juho Kangas: Konversioskripti 2.
  Konversioskripti 2:
  Luodaan uusi taulu vaccine_area rokotusalueille.
*/
CREATE TABLE vaccine_area(
  vaccine area id          NUMBER(12)    NOT NULL PRIMARY KEY,
  central hospital id      NUMBER(12)    NOT NULL,
  name                     VARCHAR2(255),
  row_timestamp            DATE          NOT NULL,
  row_creator              VARCHAR2(255) NOT NULL,
  CONSTRAINT
    fk central_hosp_id FOREIGN KEY (central_hospital_id)
  REFERENCES
    central_hospital(central_hospital_id));


/*
  20.02.2010 / Juho Kangas: Konversioskripti 3.
  Konversioskripti 1
  Luodaan uusi taulu vaccine provider, jonne tallennetaan
  rivi jokaista terveyskeskusta ja sairaalaa kohden, osoittamaan
  mihin rokotusalueeseen kyseinen laitos kuuluu.
*/
CREATE TABLE vaccine_provider(
  vaccine provider id      NUMBER(12)    NOT NULL PRIMARY KEY,
  vaccine area id          NUMBER(12)    NOT NULL,
  hospital_id              NUMBER(12),
  clinic_id                NUMBER(12),
  row timestamp            DATE          NOT NULL,
  row creator              VARCHAR2(255) NOT NULL,
  CONSTRAINT
    fk vac area FOREIGN KEY (vaccine_area_id)
  REFERENCES
    vaccine_area(vaccine_area_id),
  CONSTRAINT
    fk hosp FOREIGN KEY (hospital_id)
  REFERENCES
    hospital(hospital_id),
  CONSTRAINT
    fk_clinic FOREIGN KEY (clinic_id)
  REFERENCES
    clinic(clinic_id));


/*
  20.02.2010 / Juho Kangas: Konversioskripti 4.
  Lisätään healthcare_area-tauluun uusi kenttä central_hospital_id.
*/
ALTER TABLE healthcare area
ADD central_hospital_id  NUMBER(12)
ADD CONSTRAINT
   fk_central_hospital_id FOREIGN KEY (central_hospital_id)
REFERENCES
   central_hospital(central_hospital_id);
```

```
/*
  20.02.2010 / Juho Kangas: Konversioskripti 5.
  Konvertoidaan uuteen central_hospital-taulun sairaanhoitopiirien
  keskussairaalat. Näitä on yhteensä 20 kpl.
*/
SET SERVEROUTPUT ON;
DECLARE
  CURSOR c_central_hospitals IS
    SELECT
      h.hospital_id,
      h.name,
      h.personnel
    FROM
      hospital h
    WHERE
      h.hospital_id IN (
        SELECT
          a.hospital_id
        FROM
          area_central_hospital a);
  v count NUMBER := 0;
BEGIN
  FOR r central hospitals IN c central_hospitals LOOP
    INSERT INTO central_hospital (
      central_hospital_id,
      name,
      personnel,
      row timestamp,
      row_creator)
    VALUES(
      r_central_hospitals.hospital_id,
      r_central_hospitals.name,
      r_central_hospitals.personnel,
      sysdate,
      'Opinnäytetyö');
      v_count := v_count + 1;
  END LOOP;
  COMMIT;
  DBMS OUTPUT.PUT_LINE('Luotu ' || v_count || ' riviä');
  EXCEPTION
    WHEN OTHERS THEN
      ROLLBACK;
      DBMS_OUTPUT.PUT_LINE('Tapahtui virhe!');
END;


/*
  20.02.2010 / Juho Kangas: Konversioskripti 6.
  Konvertoidaan healthcare_area-taulun uuteen central_hospital_id-kenttään
  viittaukset keskussairaaloihin.
  Skriptissä käytetään hyväksi myöhemmin poistettavaa area_central_hospital-taulua.
*/
SET SERVEROUTPUT ON;
DECLARE
  CURSOR c_central_hospital_ids IS
    SELECT
      ach.hospital id,
      ach.healthcare_area_id
    FROM
      area_central_hospital ach;
  v_count NUMBER := 0;
BEGIN
  FOR r_central_hospital_ids IN c_central_hospital_ids LOOP
    UPDATE healthcare_area
      SET
        central_hospital_id = r_central_hospital_ids.hospital_id,
        row_timestamp = SYSDATE,
        row_creator = 'Opinnäytetyö'
      WHERE
        healthcare area id = r_central_hospital_ids.healthcare_area_id;
    v_count := v_count + 1;
  END LOOP;
  COMMIT;
  DBMS OUTPUT.PUT_LINE('Päivitetty ' || v_count || ' riviä ja hyväksytty transaktio');
  EXCEPTION
    WHEN OTHERS THEN
```

```
      ROLLBACK;
      DBMS_OUTPUT.PUT_LINE('Tapahtui virhe!');
END;


/*
  20.02.2010 / Juho Kangas: Konversioskripti 7.
  Konvertoidaan uuteen vaccine_area-taulun rokotusalueet ja
  viittaukset rokotusalueesta vastaavaan keskussairaalaan.
*/
SET SERVEROUTPUT ON;
DECLARE
  CURSOR c_central_hospital IS
    SELECT
      ch.central_hospital_id, ch.name
    FROM
      central_hospital ch
    WHERE
      name LIKE '%yliopistollinen%';
  v_count NUMBER := 0;
  v_name vaccine_area.name%TYPE;
BEGIN
  FOR r central_hospital IN c_central_hospital LOOP
    CASE
      WHEN r_central_hospital.name LIKE 'Helsingin%' THEN v_name := 'HYKS Rokotusalue';
      WHEN r_central_hospital.name LIKE 'Kuopion%' THEN v_name := 'KYS Rokotusalue';
      WHEN r_central_hospital.name LIKE 'Oulun%' THEN v_name := 'OYS Rokotusalue';
      WHEN r central_hospital.name LIKE 'Tampereen%' THEN v name := 'TAYS Rokotusalue';
      WHEN r central_hospital.name LIKE 'Turun%' THEN v_name := 'TYKS Rokotusalue';
    END CASE;
    INSERT INTO vaccine_area (
      vaccine_area_id,
      central_hospital_id,
      name,
      row timestamp,
      row_creator)
    VALUES(
      hc.nextval,
      r central_hospital.central_hospital_id,
      v name,
      SYSDATE,
      'Opinnäytetyö');
    v_count := v_count + 1;
  END LOOP;
  COMMIT;
  DBMS OUTPUT.PUT_LINE('Luotu ' || v_count || ' riviä ja hyväksytty transaktio');
  EXCEPTION
    WHEN OTHERS THEN
      ROLLBACK;
      DBMS_OUTPUT.PUT_LINE('Tapahtui virhe!');
END;


/*
  20.02.2010 / Juho Kangas: Konversioskripti 8.
  Konvertoidaan uuteen vaccine_provider-tauluun viittaukset kaikkiin
  sairaaloihin (ei keskussairaalat) ja terveyskeskuksiin sekä niiden viittaukset
  rokotusalueisiin, joihin kyseinen sairaala tai terveyskeskus kuuluu.
*/
SET SERVEROUTPUT ON;
DECLARE
  CURSOR c_hosps_and_clinics IS
    SELECT
      'H' hosp_or_clin, h.hospital_id hosp_or_clin_id, hca.name hca_name
    FROM
      hospital h, healthcare_area hca
    WHERE
      h.healthcare_area_id = hca.healthcare_area_id
      AND h.hospital_id NOT IN (SELECT central_hospital_id FROM central_hospital)
    UNION
    SELECT
      'C' hosp_or_clin, c.clinic_id hosp_or_clin_id, hca.name hca_name
    FROM
      clinic c, healthcare_area hca
    WHERE
      c.healthcare_area_id = hca.healthcare_area_id;
```

```
  v_count NUMBER := 0;
  v_hospital_id hospital.hospital_id%TYPE;
  v_hospital_name hospital.name%TYPE;
  v_hca_name healthcare_area.name%TYPE;
  v vaccine area id vaccine area.vaccine_area_id%TYPE;
  v vaccine_area vaccine_area.name%TYPE;
BEGIN
  FOR r_hosps_and_clinics IN c_hosps_and_clinics LOOP
    v_vaccine_area :=
    CASE r hosps and clinics.hca name
      WHEN 'Helsingin ja Uudenmaan sairaanhoitopiiri' THEN 'HYKS Rokotusalue'
      WHEN 'Etelä-Karjalan sairaanhoitopiiri' THEN 'HYKS Rokotusalue'
      WHEN 'Kymenlaakson sairaanhoitopiiri' THEN 'HYKS Rokotusalue'
      WHEN 'Pohjois-Savon sairaanhoitopiiri' THEN 'KYS Rokotusalue'
      WHEN 'Etelä-Savon sairaanhoitopiiri' THEN 'KYS Rokotusalue'
      WHEN 'Itä-Savon sairaanhoitopiiri' THEN 'KYS Rokotusalue'
      WHEN 'Keski-Suomen sairaanhoitopiiri' THEN 'KYS Rokotusalue'
      WHEN 'Pohjois-Karjalan sairaanhoitopiiri' THEN 'KYS Rokotusalue'
      WHEN 'Pohjois-Pohjanmaan sairaanhoitopiiri' THEN 'OYS Rokotusalue'
      WHEN 'Länsi-Pohjan sairaanhoitopiiri'  THEN 'OYS Rokotusalue'
      WHEN 'Lapin sairaanhoitopiiri' THEN 'OYS Rokotusalue'
      WHEN 'Kainuun sairaanhoitopiiri' THEN 'OYS Rokotusalue'
      WHEN 'Keski-Pohjanmaan sairaanhoitopiiri' THEN 'OYS Rokotusalue'
      WHEN 'Pirkanmaan sairaanhoitopiiri' THEN 'TAYS Rokotusalue'
      WHEN 'Etelä-Pohjanmaan sairaanhoitopiiri' THEN 'TAYS Rokotusalue'
      WHEN 'Kanta-Hämeen sairaanhoitopiiri' THEN 'TAYS Rokotusalue'
      WHEN 'Päijät-Hämeen sairaanhoitopiiri' THEN 'TAYS Rokotusalue'
      WHEN 'Vaasan sairaanhoitopiiri' THEN 'TAYS Rokotusalue'
      WHEN 'Varsinais-Suomen sairaanhoitopiiri' THEN 'TYKS Rokotusalue'
      WHEN 'Satakunnan sairaanhoitopiiri' THEN 'TYKS Rokotusalue'
    END;
    EXECUTE IMMEDIATE 'SELECT vaccine_area_id FROM vaccine_area WHERE name =
'''||v_vaccine_area||''''' INTO v_vaccine_area_id;
    IF (r hosps and clinics.hosp or clin = 'H') THEN
      INSERT INTO vaccine_provider (
        vaccine_provider_id,
        vaccine area_id,
        hospital id,
        clinic id,
        row timestamp,
        row creator)
      VALUES(
        hc.nextval,
        v vaccine area id,
        r hosps_and_clinics.hosp_or_clin_id,
        NULL,
        SYSDATE,
        'Opinnäytetyö');
      ELSE
        INSERT INTO vaccine provider (
        vaccine provider_id,
        vaccine_area_id,
        hospital_id,
        clinic_id,
        row_timestamp,
        row_creator)
      VALUES(
        hc.nextval,
        v_vaccine_area_id,
        NULL,
        r_hosps_and_clinics.hosp_or_clin_id,
        SYSDATE,
        'Opinnäytetyö');
    END IF;
    v_count := v_count + 1;
  END LOOP;
  COMMIT;
  DBMS OUTPUT.PUT_LINE('Luotu ' || v_count || ' riviä ja hyväksytty transaktio');
  EXCEPTION
    WHEN OTHERS THEN
      ROLLBACK;
      DBMS_OUTPUT.PUT_LINE('Tapahtui virhe!');
END;
```

```
/*
  20.02.2010 / Juho Kangas: Konversioskripti 9.
  Poistetaan turha area_central_hospital-taulu
*/
DROP TABLE area_central_hospital;


/*
  20.02.2010 / Juho Kangas: Konversioskripti 10.
  Lopuksi poistetaan keskussairaalat hospital-taulusta,
  jotka ovat siirretty central_hospital-tauluun.
*/
DELETE FROM hospital
WHERE hospital_id IN (SELECT central_hospital_id FROM central_hospital);
COMMIT;
```